

1 과목 소프트웨어 설계

핵심 001 소프트웨어 생명 주기 (Software Life Cycle)

소프트웨어 생명 주기는 소프트웨어 개발 방법론의 바탕이 되는 것으로, 소프트웨어를 개발하기 위해 정의하고 운용, 유지보수 등의 과정을 각 단계별로 나눈 것이다.

- 소프트웨어 생명 주기는 소프트웨어 개발 단계와 각 단계별 주요 활동, 그리고 활동의 결과에 대한 산출물로 표현한다. 소프트웨어 수명 주기라고도 한다.
- 소프트웨어 생명 주기를 표현하는 형태를 소프트웨어 생명 주기 모형이라고 하며, 소프트웨어 프로세스 모형 또는 소프트웨어 공학 패러다임이라고도 한다.

※ 소프트웨어 공학의 개념

소프트웨어 공학(SE; Software Engineering)은 소프트웨어의 위기를 극복하기 위한 방안으로 연구된 학문이며 여러 가지 방법론과 도구, 관리 기법들을 통하여 소프트웨어의 품질과 생산성 향상을 목적으로 한다.

※ 소프트웨어 공학의 기본 원칙

- 현대적인 프로그래밍 기술을 계속적으로 적용해야 한다.
- 개발된 소프트웨어의 품질이 유지되도록 지속적으로 검증해야 한다.
- 소프트웨어 개발 관련 사항 및 결과에 대한 명확한 기록을 유지해야 한다.

핵심 002 폭포수 모형(Waterfall Model)

폭포수 모형은 폭포에서 한번 떨어진 물은 거슬러 올라갈 수 없듯이 소프트웨어 개발도 이전 단계로 돌아갈 수 없다는 전제하에 각 단계를 확실히 매듭짓고 그 결과를 철저하게 검토하여 승인 과정을 거친 후에 다음 단계를 진행하는 개발 방법론이다.

- 폭포수 모형은 소프트웨어 공학에서 가장 오래되고 가장 폭넓게 사용된 전통적인 소프트웨어 생명 주기 모형으로, 고전적 생명 주기 모형이라고도 한다.

- 소프트웨어 개발 과정의 한 단계가 끝나야만 다음 단계로 넘어갈 수 있는 선형 순차적 모형이다.
- 모형을 적용한 경험과 성공 사례가 많다.
- 각 단계가 끝난 후에는 다음 단계를 수행하기 위한 결과물이 명확하게 산출되어야 한다.

불합격 방지용 안전장치 기억상자

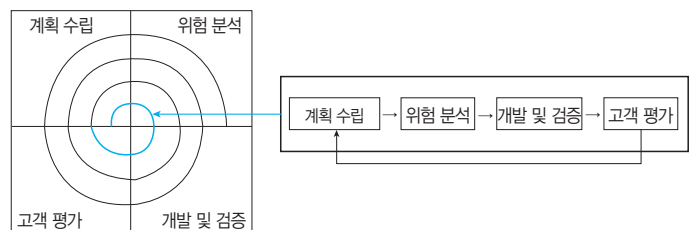
틀린 문제만 모아 오답 노트를 만들고 싶다고요?
꺼먹기 전에 다시 한 번 복습하고 싶다고요?
지금 당장 QR 코드를 스캔해 보세요.



핵심 003 나선형 모형(Spiral Model, 점진적 모형)

나선형 모형은 보헴(Boehm)이 제안한 것으로, 폭포수 모형과 프로토타입 모형의 장점에 위험 분석 기능을 추가한 모형이다.

- 나선을 따라 돌듯이 여러 번의 소프트웨어 개발 과정을 거쳐 점진적으로 완벽한 최종 소프트웨어를 개발하는 것으로, 점진적 모형이라고도 한다.
- 소프트웨어를 개발하면서 발생할 수 있는 위험을 관리하고 최소화하는 것을 목적으로 한다.
- 점진적으로 개발 과정이 반복되므로 누락되거나 추가된 요구사항을 첨가할 수 있고, 정밀하며, 유지보수 과정이 필요 없다.



핵심 004 애자일 모형(Agile Model)

애자일은 '민첩한', '기민한'이라는 의미로, 고객의 요구사항 변화에 유연하게 대응할 수 있도록 일정한 주기를 반복하면서 개발과정을 진행한다.

- 애자일 모형은 어느 특정 개발 방법론이 아니라 좋은 것을 빠르고 낭비 없게 만들기 위해 고객과의 소통에 초점을 맞춘 방법론을 통칭한다.

- 애자일 모형은 기업 활동 전반에 걸쳐 사용된다.
- 애자일 모형을 기반으로 하는 소프트웨어 개발 모형에는 스크럼(Scrum), XP(eXtreme Programming), 칸반(Kanban), Lean, 크리스탈(Crystal), ASD(Adaptive Software Development), 기능 중심 개발(FDD; Feature Driven Development), DSDM(Dynamic System Development Method), DAD(Disciplined Agile Delivery) 등이 있다.

※ 애자일 개발 4가지 핵심 가치

1. 프로세스와 도구보다는 개인과 상호작용에 더 가치를 둔다.
2. 방대한 문서보다는 실행되는 SW에 더 가치를 둔다.
3. 계약 협상보다는 고객과 협업에 더 가치를 둔다.
4. 계획을 따르기 보다는 변화에 반응하는 것에 더 가치를 둔다.

핵심 005 스크럼의 개요

스크럼이란 럭비에서 반칙으로 경기가 중단된 경우 양팀의 선수들이 럭비공을 가운데 두고 상대팀을 밀치기 위해 서로 대치해 있는 대형을 말한다. 스크럼은 이처럼 팀이 중심이 되어 개발의 효율성을 높인다는 의미가 내포된 용어이다.

- 스크럼은 팀원 스스로가 스크럼 팀을 구성(self-organizing)해야 하며, 개발 작업에 관한 모든 것을 스스로 해결(cross-functional)할 수 있어야 한다.
- 스크럼 팀은 제품 책임자, 스크럼 마스터, 개발팀으로 구성된다.

제품 책임자(PO; Product Owner)

- 이해관계자들 중 개발될 제품에 대한 이해도가 높고, 요구사항을 책임지고 의사 결정할 사람으로 선정하는데, 주로 개발 의뢰자나 사용자가 담당한다.
- 이해관계자들의 의견을 종합하여 제품에 대한 요구사항을 작성하는 주체다.
- 제품에 대한 테스트를 수행하면서 주기적으로 요구사항의 우선순위를 갱신한다.

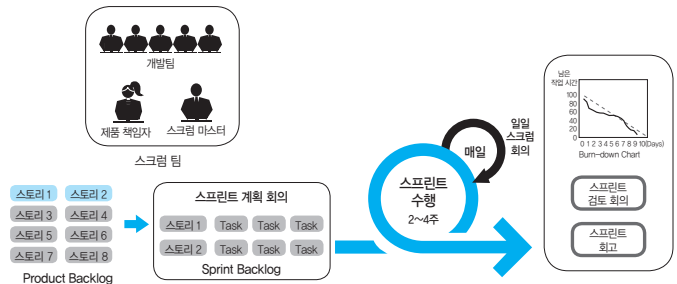
스크럼 마스터(SM; Scrum Master)

- 스크럼 팀이 스크럼을 잘 수행할 수 있도록 객관적인 시각에서 조언을 해주는 가이드 역할을 수행한다. 팀원들을 통제하는 것이 목표가 아니다.
- 일일 스크럼 회의를 주관하여 진행 사항을 점검하고, 개발 과정에서 발생된 장애 요소를 공론화하여 처리한다.

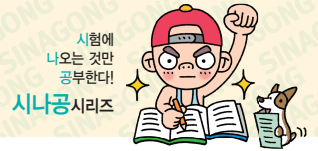
개발팀(DT; Development Team)

- 제품 책임자와 스크럼 마스터를 제외한 모든 팀원으로, 개발자 외에도 디자이너, 테스터 등 제품 개발을 위해 참여하는 모든 사람이 대상이 된다.
- 보통 최대 인원은 7~8명이 적당하다.

핵심 006 스크럼 개발 프로세스



제품 백로그 (Product Backlog)	제품 개발에 필요한 모든 요구사항(User Story)을 우선순위에 따라 나열한 목록이다.
스프린트 계획 회의 (Sprint Planning Meeting)	제품 백로그 중 이번 스프린트에서 수행할 작업을 대상으로 단기 일정을 수립하는 것이다.
스프린트(Sprint)	실제 개발 작업을 진행하는 과정으로, 보통 2~4주 정도의 기간 내에서 진행한다.
일일 스크럼 회의 (Daily Scrum Meeting)	<ul style="list-style-type: none"> • 모든 팀원이 매일 약속된 시간에 약 15분 정도의 짧은 시간동안 진행 상황을 점검한다. • 회의는 보통 서서 진행하며, 남은 작업 시간은 소멸 차트(Burn-down Chart)에 표시한다.
스프린트 검토 회의 (Sprint Review)	부분 또는 전체 완성 제품이 요구사항에 잘 부합되는지 사용자가 포함된 참석자 앞에서 테스트를 수행한다.
스프린트 회고 (Sprint Retrospective)	스프린트 주기를 되돌아보며 정해놓은 규칙을 잘 준수했는지, 개선할 점은 없는지 등을 확인하고 기록한다.



핵심

007

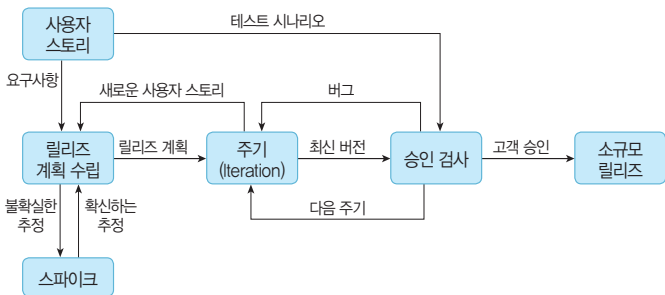
XP(eXtreme Programming) 및 개발 프로세스

XP(eXtreme Programming)

XP(eXtreme Programming)는 수시로 발생하는 고객의 요구사항에 유연하게 대응하기 위해 고객의 참여와 개발 과정의 반복을 극대화하여 개발 생산성을 향상시키는 방법이다.

- XP는 짧고 반복적인 개발 주기, 단순한 설계, 고객의 적극적인 참여를 통해 소프트웨어를 빠르게 개발하는 것을 목적으로 한다.
- 릴리즈의 기간을 짧게 반복하면서 고객의 요구사항 반영에 대한 가시성을 높인다.
- XP의 5가지 핵심 가치: 의사소통(Communication), 단순성(Simplicity), 용기(Courage), 존중(Respect), 피드백(Feedback)

XP 개발 프로세스



사용자 스토리 (User Story)	<ul style="list-style-type: none"> • 고객의 요구사항을 간단한 시나리오로 표현한 것이다. • 내용은 기능 단위로 구성하며, 필요한 경우 간단한 테스트 사항(Test Case)도 기재한다.
릴리즈 계획 수립(Release Planning)	<ul style="list-style-type: none"> • 몇 개의 스토리가 적용되어 부분적으로 기능이 완료된 제품을 제공하는 것을 릴리즈라고 한다. • 부분 혹은 전체 개발 완료 시점에 대한 일정을 수립한다.
스파이크 (Spike)	요구사항의 신뢰성을 높이고 기술 문제에 대한 위험을 감소시키기 위해 별도로 만드는 간단한 프로그램이다.
이테레이션 (Iteration)	<ul style="list-style-type: none"> • 하나의 릴리즈를 더 세분화 한 단위를 이테레이션(iteration)이라고 한다. • 일반적으로 1~3주 정도의 기간으로 진행된다.
승인 검사 (Acceptance Test, 인수 테스트)	<ul style="list-style-type: none"> • 하나의 이테레이션 안에서 계획된 릴리즈 단위의 부분 완료 제품이 구현되면 수행하는 테스트이다. • 테스트가 완료되면 다음 이테레이션을 진행한다.

소규모 릴리즈 (Small Release)

릴리즈를 소규모로 하게 되면, 고객의 반응을 기능 별로 확인할 수 있어, 고객의 요구사항에 좀 더 유연하게 대응할 수 있다.

핵심

008

XP의 주요 실천 방법(Practice)

Pair Programming (짝 프로그래밍)	다른 사람과 함께 프로그래밍을 수행함으로써 개발에 대한 책임을 공동으로 나눠 갖는 환경을 조성한다.
Collective Ownership (공동 코드 소유)	개발 코드에 대한 권한과 책임을 공동으로 소유한다.
Test-Driven Development (테스트 주도 개발)	<ul style="list-style-type: none"> • 개발자가 실제 코드를 작성하기 전에 테스트 케이스를 먼저 작성하므로 자신이 무엇을 해야할지를 정확히 파악한다. • 테스트가 지속적으로 진행될 수 있도록 자동화된 테스트 도구(구조, 프레임워크)를 사용한다.
Whole Team (전체 팀)	개발에 참여하는 모든 구성원(고객 포함)들은 각자 자신의 역할이 있고 그 역할에 대한 책임을 가져야 한다.
Continuous Integration (계속적인 통합)	모듈 단위로 나눠서 개발된 코드들은 하나의 작업이 마무리될 때마다 지속적으로 통합된다.
Design Improvement (디자인 개선) 또는 Refactoring(리팩토링)	프로그램 기능의 변경 없이, 단순화, 유연성 강화 등을 통해 시스템을 재구성한다.
Small Releases (소규모 릴리즈)	릴리즈 기간을 짧게 반복함으로써 고객의 요구 변화에 신속히 대응할 수 있다.

핵심

009

현행 시스템 파악

단계	현행 시스템	내용
1단계	시스템 구성 파악	조직의 주요 업무를 담당하는 기간 업무와 이를 지원하는 지원 업무로 구분하여 기술함
	시스템 기능 파악	현재 제공하는 기능들을 주요 기능과 하부 기능, 세부 기능으로 구분하여 계층형으로 표시함
	시스템 인터페이스 파악	단위 업무 시스템 간에 주고받는 데이터의 종류, 형식, 프로토콜, 연계 유형, 주기 등을 명시함

2단계	아키텍처 구성 파악	최상위 수준에서 계층별로 표현한 아키텍처 구성도를 작성함
	소프트웨어 구성 파악	소프트웨어들의 제품명, 용도, 라이선스 적용 방식, 라이선스 수 등을 명시함
3단계	하드웨어 구성 파악	단위 업무 시스템들이 운용되는 서버의 주요 사양과 수량, 그리고 서버의 이중화의 적용 여부를 명시함
	네트워크 구성 파악	서버의 위치, 서버 간의 네트워크 연결 방식을 네트워크 구성도로 작성함

핵심 010 운영체제(OS, Operating System)

운영체제는 컴퓨터 시스템의 자원들을 효율적으로 관리하며, 사용자가 컴퓨터를 편리하고 효율적으로 사용할 수 있도록 환경을 제공하는 소프트웨어이다.

- 컴퓨터 사용자와 컴퓨터 하드웨어 간의 인터페이스로서 동작하는 시스템 소프트웨어의 일종으로, 다른 응용 프로그램이 유용한 작업을 할 수 있도록 환경을 제공해준다.
- 컴퓨터 운영체제의 종류에는 Windows, UNIX, Linux, Mac OS 등이, 모바일 운영체제에는 iOS, Android 등이 있다.
- 운영체제 관련 요구사항 식별 시 고려사항
 - 가용성
 - 성능
 - 기술 지원
 - 주변 기기
 - 구축 비용

핵심 011 데이터베이스 관리 시스템(DBMS)

DBMS(DataBase Management System)는 사용자와 데이터베이스 사이에서 사용자의 요구에 따라 정보를 생성해 주고, 데이터베이스를 관리해 주는 소프트웨어이다.

- DBMS는 기존의 파일 시스템이 갖는 데이터의 종속성과 중복성의 문제를 해결하기 위해 제안된 시스템으로, 모든 응용 프로그램들이 데이터베이스를 공유할 수 있도록 관리해 준다.

- DBMS는 데이터베이스의 구성, 접근 방법, 유지관리에 대한 모든 책임을 진다.
- DBMS의 종류에는 Oracle, IBM DB2, Microsoft SQL Server, MySQL, SQLite, MongoDB, Redis 등이 있다.
- DBMS 관련 요구사항 식별 시 고려사항
 - 가용성
 - 성능
 - 기술 지원
 - 상호 호환성
 - 구축 비용

핵심 012 요구사항의 개념 / 요구사항의 유형

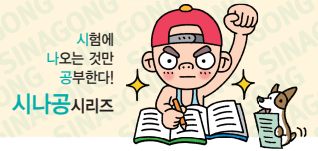
요구사항은 소프트웨어가 어떤 문제를 해결하기 위해 제공하는 서비스에 대한 설명과 정상적으로 운영되는데 필요한 제약조건 등을 나타낸다.

- 요구사항은 소프트웨어 개발이나 유지 보수 과정에서 필요한 기준과 근거를 제공한다.

요구사항의 유형

요구사항은 일반적으로 기술하는 내용에 따라 기능 요구사항(Functional requirements)과 비기능 요구사항(Non-functional requirements)으로 구분하며, 기술 관점과 대상의 범위에 따라 시스템 요구사항(System requirements)과 사용자 요구사항(User requirements)으로 나눈다.

유형	내용
기능 요구사항 (Functional requirements)	<ul style="list-style-type: none"> • 시스템이 무엇을 하는지, 어떤 기능을 하는지에 대한 사항 • 시스템의 입력이나 출력으로 무엇이 포함되어야 하는지, 시스템이 어떤 데이터를 저장하거나 연산을 수행해야 하는지에 대한 사항 • 시스템이 반드시 수행해야 하는 기능 • 사용자가 시스템을 통해 제공받기를 원하는 기능



비기능 요구사항 (Non-functional requirements)

- 시스템 장비 구성 요구사항 : 하드웨어, 소프트웨어, 네트워크 등의 시스템 장비 구성에 대한 요구사항
- 성능 요구사항 : 처리 속도 및 시간, 처리량, 동적·정적 적용량, 가용성 등 성능에 대한 요구사항
- 인터페이스 요구사항 : 시스템 인터페이스와 사용자 인터페이스에 대한 요구사항으로 다른 소프트웨어, 하드웨어 및 통신 인터페이스, 다른 시스템과의 정보 교환에 사용되는 프로토콜과의 연계도 포함하여 기술
- 데이터 요구사항 : 초기 자료 구축 및 데이터 변환을 위한 대상, 방법, 보안이 필요한 데이터 등 데이터를 구축하기 위해 필요한 요구사항
- 테스트 요구사항 : 도입되는 장비의 성능 테스트(BMT)나 구축된 시스템이 제대로 운영되는지를 테스트하고 점검하기 위한 테스트 요구사항
- 보안 요구사항 : 시스템의 데이터 및 기능, 운영 접근을 통제하기 위한 요구사항
- 품질 요구사항 : 관리가 필요한 품질 항목, 품질 평가 대상에 대한 요구사항으로 가용성, 정합성, 상호 호환성, 대응성, 신뢰성, 사용성, 유지·관리성, 이식성, 확장성, 보안성 등으로 구분하여 기술
- 제약사항 : 시스템 설계, 구축, 운영과 관련하여 사전에 파악된 기술, 표준, 업무, 법·제도 등의 제약조건
- 프로젝트 관리 요구사항 : 프로젝트의 원활한 수행을 위한 관리 방법에 대한 요구사항
- 프로젝트 지원 요구사항 : 프로젝트의 원활한 수행을 위한 지원 사항이나 방안에 대한 요구사항

- 요구사항을 도출하는 주요 기법에는 청취와 인터뷰, 설문, 브레인스토밍, 워크샵, 프로토타이핑, 유스케이스 등이 있다.

요구사항 분석(Requirement Analysis)

요구사항 분석은 개발 대상에 대한 사용자의 요구사항 중 명확하지 않거나 모호하여 이해되지 않는 부분을 발견하고 이를 정리하기 위한 과정이다.

- 요구사항 분석에는 자료 흐름도(DFD), 자료 사전(DD) 등의 도구가 사용된다.

요구사항 명세(Requirement Specification)

요구사항 명세는 분석된 요구사항을 바탕으로 모델을 작성하고 문서화하는 것을 의미한다.

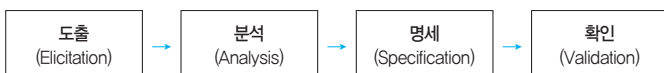
- 구체적인 명세를 위해 소단위 명세서(Mini-Spec)가 사용될 수 있다.

요구사항 확인(Requirement Validation, 요구사항 검증)

요구사항 확인은 개발 자원을 요구사항에 할당하기 전에 요구사항 명세서가 정확하고 완전하게 작성되었는지를 검토하는 활동이다.

핵심 013 요구사항 개발 프로세스

요구사항 개발 프로세스는 개발 대상에 대한 요구사항을 체계적으로 도출하고 이를 분석한 후 분석 결과를 명세서(Specification Document)에 정리한 다음 마지막으로 이를 확인 및 검증하는 일련의 구조화된 활동이다.

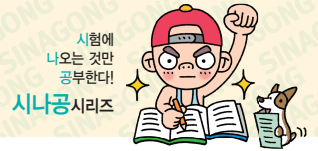


요구사항 도출(Requirement Elicitation, 요구사항 수집)

요구사항 도출은 시스템, 사용자, 그리고 시스템 개발에 관련된 사람들이 서로 의견을 교환하여 요구사항이 어디에 있는지, 어떻게 수집할 것인지를 식별하고 이해하는 과정이다.

핵심 014 요구사항 명세 기법

구분	정형 명세 기법	비정형 명세 기법
기법	수학적 원리 기반, 모델 기반	상태/기능/객체 중심
작성 방법	수학적 기호, 정형화된 표기법	일반 명사, 동사 등의 자연어를 기반으로 서술 또는 다이어그램으로 작성
특징	<ul style="list-style-type: none"> • 요구사항을 정확하고 간결하게 표현할 수 있음 • 요구사항에 대한 결과가 작성자에 관계없이 일관성이 있으므로 완전성 검증이 가능함 • 표기법이 어려워 사용자가 이해하기 어려움 	<ul style="list-style-type: none"> • 자연어의 사용으로 인해 요구사항에 대한 결과가 작성자에 따라 다를 수 있어 일관성이 떨어지고, 해석이 달라질 수 있음 • 내용의 이해가 쉬어 의사소통이 용이함
종류	VDM, Z, Petri-net, CSP 등	FSM, Decision Table, ER모델링, State Chart(SADT) 등



핵심 015 요구사항 분석의 개요

요구사항 분석은 소프트웨어 개발의 실제적인 첫 단계로 개발 대상에 대한 사용자의 요구사항을 이해하고 문서화(명세화)하는 활동을 의미한다.

- 사용자 요구의 타당성을 조사하고 비용과 일정에 대한 제약을 설정한다.
- 사용자의 요구를 정확하게 추출하여 목표를 정하고, 어떤 방식으로 해결할 것인지를 결정한다.
- 요구사항 분석을 통한 결과는 소프트웨어 설계 단계에서 필요한 기본적인 자료가 되므로 사용자의 요구사항을 정확하고 일관성 있게 분석하여 문서화해야 한다.
- 소프트웨어 분석가에 의해 요구사항 분석이 수행되며, 이 작업 단계를 요구사항 분석 단계라고 한다.

핵심 016 구조적 분석 기법

구조적 분석 기법은 자료의 흐름과 처리를 중심으로 하는 요구사항 분석 방법으로, 다음과 같은 특징이 있다.

- 도형 중심의 분석용 도구와 분석 절차를 이용하여 사용자의 요구사항을 파악하고 문서화한다.
- 도형 중심의 도구를 사용하므로 분석가와 사용자 간의 대화가 용이하다.
- 구조적 분석 기법에서는 자료 흐름도(DFD), 자료 사전(DD), 소단위 명세서(Mini-Spec.), 개체 관계도(ERD), 상태 전이도(STD), 제어 명세서 등의 도구를 이용하여 모델링한다.

핵심 017 자료 흐름도(DFD)

자료 흐름도(DFD; Data Flow Diagram)는 요구사항 분석에서 자료의 흐름 및 변환 과정과 기능을 도형 중심으로 기술하는 방법으로 자료 흐름 그래프, 버블 차트라고도 한다.

- 자료 흐름도에서는 자료의 흐름과 기능을 프로세스(Process), 자료 흐름(Flow), 자료 저장소(Data Store), 단말(Terminator)의 네 가지 기본 기호로 표시한다.

기 호	의 미	표기법	
		Yourdon/DeMarco	Gane/Sarson
프로세스 (Process)	<ul style="list-style-type: none"> • 자료를 변환시키는 시스템의 한 부분(처리 과정)을 나타내며 처리, 기능, 변환, 버블이라고도 한다. • 원이나 둥근 사각형으로 표시하고 그 안에 프로세스 이름을 기입한다. 		
자료 흐름 (Data Flow)	<ul style="list-style-type: none"> • 자료의 이동(흐름)이나 연관 관계를 나타낸다. • 화살표 위에 자료의 이름을 기입한다. 		
자료 저장소 (Data Store)	<ul style="list-style-type: none"> • 시스템에서의 자료 저장소(파일, 데이터베이스)를 나타낸다. • 도형 안에 자료 저장소 이름을 기입한다. 		
단말 (Terminator)	<ul style="list-style-type: none"> • 시스템과 교신하는 외부 개체로, 입력 데이터가 만들어지고 출력 데이터를 받는다(정보의 생산자와 소비자). • 도형 안에 이름을 기입한다. 		

핵심 018 자료 사전

자료 사전(DD; Data Dictionary)은 자료 흐름도에 있는 자료를 더 자세히 정의하고 기록한 것이며, 이처럼 데이터를 설명하는 데이터를 데이터의 데이터 또는 메타 데이터(Meta Data)라고 한다.

- 자료 흐름도에 시각적으로 표시된 자료에 대한 정보를 체계적이고 조직적으로 모아 개발자나 사용자가 편리하게 사용할 수 있다.

기 호	의 미
=	자료의 정의: ~로 구성되어 있다(is composed of)
+	자료의 연결: 그리고(and)
()	자료의 생략: 생략 가능한 자료(Optional)
[]	자료의 선택: 또는(or)
{ }	자료의 반복: Iteration of ① { } _n : n번 이상 반복 ② { } ⁿ : 최대 n번 반복 ③ { } _m ⁿ : m 이상 n 이하로 반복
* *	자료의 설명: 주석(Comment)

핵심

019

요구사항 분석을 위한 CASE (자동화 도구)

요구사항 분석을 위한 자동화 도구는 요구사항을 자동으로 분석하고, 요구사항 분석 명세서를 기술하도록 개발된 도구를 의미한다.

종류

- SADT(Structured Analysis and Design Technique) : SoftTech사에서 개발한 것으로 시스템 정의, 소프트웨어 요구사항 분석, 시스템/소프트웨어 설계를 위해 널리 이용되어 온 구조적 분석 및 설계 도구이다.
- SREM(Software Requirements Engineering Methodology) = RSL/REVS
 - TRW사가 우주 국방 시스템 그룹에 의해 실시간 처리 소프트웨어 시스템에서 요구사항을 명확히 기술하도록 할 목적으로 개발한 것으로, RSL과 REVS를 사용하는 자동화 도구이다.
 - RSL(Requirement Statement Language) : 요소, 속성, 관계, 구조들을 기술하는 요구사항 기술 언어
 - REVS(Requirement Engineering and Validation System) : RSL로 기술된 요구사항들을 자동으로 분석하여 요구사항 분석 명세서를 출력하는 요구사항 분석기
- PSL/PSA : 미시간 대학에서 개발한 것으로 PSL과 PSA를 사용하는 자동화 도구이다.
- TAGS(Technology for Automated Generation of Systems) : 시스템 공학 방법 응용에 대한 자동 접근 방법으로, 개발 주기의 전 과정에 이용할 수 있는 통합 자동화 도구이다.

핵심

020

HIPO

HIPO(Hierarchy Input Process Output)는 시스템의 분석 및 설계나 문서화할 때 사용되는 기법으로, 시스템 실행 과정인 입력, 처리, 출력의 기능을 나타낸다.

- 기본 시스템 모델은 입력, 처리, 출력으로 구성되며, 하향식 소프트웨어 개발을 위한 문서화 도구이다.

- 체계적인 문서 관리가 가능하다.
- 기호, 도표 등을 사용하므로 보기 쉽고 이해하기도 쉽다.
- 기능과 자료의 의존 관계를 동시에 표현할 수 있다.
- 변경, 유지보수가 용이하다.
- 시스템의 기능을 여러 개의 고유 모듈들로 분할하여 이들 간의 인터페이스를 계층 구조로 표현한 것을 HIPO Chart라고 한다.

HIPO Chart의 종류

- 가시적 도표(도식 목차) : 시스템의 전체적인 기능과 흐름을 보여주는 계층(Tree) 구조도
- 총체적 도표(총괄도표, 개요 도표) : 프로그램을 구성하는 기능을 기술한 것으로 입력, 처리, 출력에 대한 전반적인 정보를 제공하는 도표
- 세부적 도표(상세 도표) : 총체적 도표에 표시된 기능을 구성하는 기본 요소들을 상세히 기술하는 도표

핵심

021

UML(Unified Modeling Language)의 개요

UML은 시스템 분석, 설계, 구현 등 시스템 개발 과정에서 시스템 개발자와 고객 또는 개발자 상호간의 의사소통이 원활하게 이루어지도록 표준화한 대표적인 객체지향 모델링 언어이다.

- UML은 Rumbaugh(OMT), Booch, Jacobson 등의 객체지향 방법론의 장점을 통합하였으며, 객체 기술에 관한 국제표준화기구인 OMG(Object Management Group)에서 표준으로 지정하였다.
- UML을 이용하여 시스템의 구조를 표현하는 6개의 구조 다이어그램과 시스템의 동작을 표현하는 7개의 행위 다이어그램을 작성할 수 있다.
- 각각의 다이어그램은 사물과 사물 간의 관계를 용도에 맞게 표현한다.
- UML의 구성 요소에는 사물(Things), 관계(Relationships), 다이어그램(Diagram) 등이 있다.

핵심 022 관계(Relationships)

관계는 사물과 사물 사이의 연관성을 표현하는 것으로, 연관 관계, 집합 관계, 포함 관계, 일반화 관계, 의존 관계, 실체화 관계 등이 있다.

연관(Association) 관계

연관 관계는 2개 이상의 사물이 서로 관련되어 있음을 표현한다.

예제1 사람이 집을 소유하는 관계이다. 사람은 자기가 소유하고 있는 집에 대해 알고 있지만 집은 누구에 의해 자신이 소유되고 있는지 모른다는 의미이다.



해설

- ‘사람’ 쪽에 표기된 다중도가 ‘1’이므로 집은 한 사람에게 의해서만 소유될 수 있다.
- ‘집’ 쪽에 표기된 다중도가 ‘1’이므로 사람은 집을 하나만 소유할 수 있다.

예제2 선생님은 학생을 가르치고 학생은 선생님으로부터 가르침을 받는 것과 같이 선생님과 학생은 서로 관계가 있다.



해설

- ‘선생님’ 쪽에 표기된 다중도가 ‘1..*’이므로 학생은 한 명 이상의 선생님으로부터 가르침을 받는다.
- ‘학생’ 쪽에 표기된 다중도가 ‘1..*’이므로 선생님은 한 명 이상의 학생을 가르친다.

집합(Aggregation) 관계

집합 관계는 하나의 사물이 다른 사물에 포함되어 있는 관계를 표현한다.

예제 프린터는 컴퓨터에 연결해서 사용할 수 있으며, 다른 컴퓨터에 연결해서 사용할 수도 있다.



포함(Composition) 관계

포함 관계는 집합 관계의 특수한 형태로, 포함하는 사물의 변화가 포함되는 사물에게 영향을 미치는 관계를 표현한다.

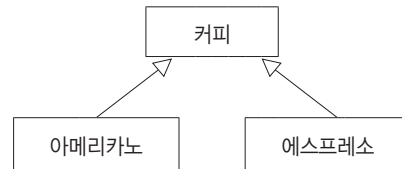
예제 문을 열 수 있는 키는 하나이며, 해당 키로 다른 문은 열 수 없다. 문이 없어도 키도 더 이상 필요하지 않다.



일반화(Generalization) 관계

일반화 관계는 하나의 사물이 다른 사물에 비해 더 일반적인지 구체적인지를 표현한다.

예제 아메리카노와 에스프레소는 커피이다. 다시 말하면, 커피에는 아메리카노와 에스프레소가 있다.



의존(Dependency) 관계

의존 관계는 연관 관계와 같이 사물 사이에 서로 연관 있으나 필요에 의해 서로에게 영향을 주는 짧은 시간 동안만 연관을 유지하는 관계를 표현한다.

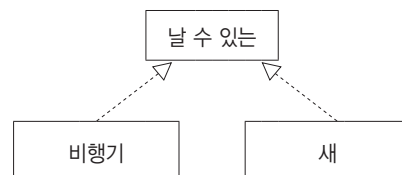
예제 등급이 높으면 할인율을 적용하고, 등급이 낮으면 할인율을 적용하지 않는다.

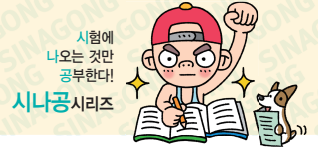


실체화(Realization) 관계

실체화 관계는 사물이 할 수 있거나 해야 하는 기능(행위, 인터페이스)으로 서로를 그룹화 할 수 있는 관계를 표현한다.

예제 비행기는 날 수 있고 새도 날 수 있다. 그러므로 비행기와 새는 날 수 있다는 행위로 그룹화 할 수 있다.





핵심 023 다이어그램(Diagram)

다이어그램은 사물과 관계를 도형으로 표현한 것이다.

- 여러 관점에서 시스템을 가시화한 뷰(View)를 제공함으로써 의사소통에 도움을 준다.
- 정적 모델링에서는 주로 구조적 다이어그램을 사용하고 동적 모델링에서는 주로 행위 다이어그램을 사용한다.
- 구조적(Structural) 다이어그램의 종류

클래스 다이어그램 (Class Diagram)	<ul style="list-style-type: none"> • 클래스와 클래스가 가지는 속성, 클래스 사이의 관계를 표현한다. • 시스템의 구조를 파악하고 구조상의 문제점을 도출할 수 있다.
객체 다이어그램 (Object Diagram)	<ul style="list-style-type: none"> • 클래스에 속한 사물(객체)들, 즉 인스턴스(Instance)를 특정 시점의 객체와 객체 사이의 관계로 표현한다. • 럼바우(Rumbaugh) 객체지향 분석 기법에서 객체 모델링에 활용된다.
컴포넌트 다이어그램 (Component Diagram)	<ul style="list-style-type: none"> • 실제 구현 모듈인 컴포넌트 간의 관계나 컴포넌트 간의 인터페이스를 표현한다. • 구현 단계에서 사용되는 다이어그램이다.
배치 다이어그램 (Deployment Diagram)	<ul style="list-style-type: none"> • 결과물, 프로세스, 컴포넌트 등 물리적 요소들의 위치를 표현한다. • 노드와 의사소통(통신) 경로로 표현한다. • 구현 단계에서 사용되는 다이어그램이다.
복합체 구조 다이어그램 (Composite Structure Diagram)	클래스나 컴포넌트가 복합 구조를 갖는 경우 그 내부 구조를 표현한다.
패키지 다이어그램 (Package Diagram)	유스케이스나 클래스 등의 모델 요소들을 그룹화한 패키지들의 관계를 표현한다.

- 행위(Behavioral) 다이어그램의 종류

유스케이스 다이어그램 (Use Case Diagram)	<ul style="list-style-type: none"> • 사용자의 요구를 분석하는 것으로 기능 모델링 작업에 사용한다. • 사용자(Actor)와 사용 사례(Use Case)로 구성되며, 사용 사례 간에는 여러 형태의 관계로 이루어진다.
시퀀스 다이어그램 (Sequence Diagram)	상호 작용하는 시스템이나 객체들이 주고받는 메시지를 표현한다.

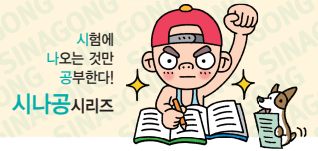
커뮤니케이션 다이어그램 (Communication Diagram)	시퀀스 다이어그램과 같이 동작에 참여하는 객체들이 주고받는 메시지를 표현하는데, 메시지뿐만 아니라 객체들 간의 연관까지 표현한다.
상태 다이어그램 (State Diagram)	<ul style="list-style-type: none"> • 하나의 객체가 자신이 속한 클래스의 상태 변화 혹은 다른 객체와의 상호 작용에 따라 상태가 어떻게 변화하는지를 표현한다. • 럼바우(Rumbaugh) 객체지향 분석 기법에서 동적 모델링에 활용된다.
활동 다이어그램 (Activity Diagram)	시스템이 어떤 기능을 수행하는지 객체의 처리 로직이나 조건에 따른 처리의 흐름을 순서에 따라 표현한다.
상호작용 개요 다이어그램 (Interaction Overview Diagram)	상호작용 다이어그램 간의 제어 흐름을 표현한다.
타이밍 다이어그램 (Timing Diagram)	객체 상태 변화와 시간 제약을 명시적으로 표현한다.

핵심 024 스테레오 타입(Stereotype)

스테레오 타입은 UML에서 표현하는 기본 기능 외에 추가적인 기능을 표현하기 위해 사용한다.

- 길러멧(Guilemet)이라고 부르는 겹화살괄호(《 》) 사이에 표현할 형태를 기술한다.
- 주로 표현되는 형태는 다음과 같다.

《include》	연결된 다른 UML 요소에 대해 포함 관계에 있는 경우
《extend》	연결된 다른 UML 요소에 대해 확장 관계에 있는 경우
《interface》	인터페이스를 정의하는 경우
《exception》	예외를 정의하는 경우
《constructor》	생성자 역할을 수행하는 경우



핵심 025 시퀀스(Sequence) 다이어그램

시퀀스 다이어그램은 시스템이나 객체들이 메시지를 주고받으며 시간의 흐름에 따라 상호 작용하는 과정을 액터, 객체, 메시지 등의 요소를 사용하여 그림으로 표현한 것이다.

시퀀스 다이어그램의 구성 요소

시퀀스 다이어그램은 액터, 객체, 생명선, 실행, 메시지 등으로 구성된다.

액터(Actor)	시스템으로부터 서비스를 요청하는 외부 요소로, 사람이나 외부 시스템을 의미함
객체(Object)	메시지를 주고받는 주체
생명선(Lifeline)	객체가 메모리에 존재하는 기간으로, 객체 아래쪽에 점선을 그어 표현함
실행 상자(Active Box)	객체가 메시지를 주고받으며 구동되고 있음을 표현함
메시지(Message)	객체가 상호 작용을 위해 주고받는 메시지

핵심 026 사용자 인터페이스(UI, User Interface)

- 사용자 인터페이스(UI)는 사용자와 시스템 간의 상호 작용이 원활하게 이뤄지도록 도와주는 장치나 소프트웨어를 의미한다.
- 사용자의 만족도에 가장 큰 영향을 미치는 중요한 요소로, 소프트웨어 영역 중 변경이 가장 많이 발생한다.
- 사용자의 편리성과 가독성을 높임으로써 작업 시간을 단축시키고 업무에 대한 이해도를 높여준다.
- 최소한의 노력으로 원하는 결과를 얻을 수 있게 한다.
- 사용자 인터페이스의 기본 원칙
 - 직관성 : 누구나 쉽게 이해하고 사용할 수 있어야 한다.
 - 유효성 : 사용자의 목적을 정확하고 완벽하게 달성해야 한다.
 - 학습성 : 누구나 쉽게 배우고 익힐 수 있어야 한다.
 - 유연성 : 사용자의 요구사항을 최대한 수용하고 실수를 최소화해야 한다.

- 사용자 인터페이스 개발 시스템의 기능
 - 사용자의 입력을 검증할 수 있어야 한다.
 - 에러 처리와 그와 관련된 에러 메시지를 표시할 수 있어야 한다.
 - 도움과 프롬프트(Prompt)를 제공해야 한다.

핵심 027 UI 설계 도구

UI 설계 도구

UI 설계 도구는 사용자의 요구사항에 맞게 UI의 화면 구조나 화면 배치 등을 설계할 때 사용하는 도구로, 종류에는 와이어프레임, 목업, 스토리보드, 프로토타입, 유스케이스 등이 있다.

와이어프레임(Wireframe)

- 와이어프레임은 기획 단계의 초기에 제작하는 것으로, 페이지에 대한 개략적인 레이아웃이나 UI 요소 등에 대한 뼈대를 설계하는 단계이다.
- 개발자나 디자이너 등이 레이아웃을 협의하거나 현재 진행 상태 등을 공유하기 위해 와이어프레임을 사용한다.
- 와이어프레임 툴 : 손그림, 파워포인트, 키노트, 스케치, 일러스트, 포토샵 등

목업(Mockup)

- 목업은 디자인, 사용 방법 설명, 평가 등을 위해 와이어프레임보다 좀 더 실제 화면과 유사하게 만든 정적인 형태의 모형이다.
- 시각적으로만 구성 요소를 배치하는 것으로 실제로 구현되지는 않는다.
- 목업 툴 : 파워 목업, 발사믹 목업 등

스토리보드(Story Board)

- 스토리보드는 와이어프레임에 콘텐츠에 대한 설명, 페이지 간 이동 흐름 등을 추가한 문서이다.
- 디자이너와 개발자가 최종적으로 참고하는 작업 지침서로, 정책, 프로세스, 콘텐츠 구성, 와이어프레임, 기능 정의 등 서비스 구축을 위한 모든 정보가 들어 있다.
- 스토리보드 툴 : 파워포인트, 키노트, 스케치, Axure 등

프로토타입(Prototype)

- 프로토타입은 와이어프레임이나 스토리보드 등에 인터랙션을 적용함으로써 실제 구현된 것처럼 테스트가 가능한 동적인 형태의 모형이다.
- 프로토타입은 사용성 테스트나 작업자 간 서비스 이해를 위해 작성하는 샘플이다.

유스케이스(Use Case)

- 유스케이스는 사용자 측면에서의 요구사항으로, 사용자가 원하는 목표를 달성하기 위해 수행할 내용을 기술한다.
- 사용자의 요구사항을 빠르게 파악함으로써 프로젝트의 초기에 시스템의 기능적인 요구를 결정하고 그 결과를 문서화할 수 있다.

• UI 요구사항 작성 순서

요구사항 요소 확인	파악된 요구사항 요소의 종류와 각각의 표현 방식 등을 검토한다. • 요구사항 요소 : 데이터 요구, 기능 요구, 제품/서비스의 품질, 제약 사항
정황 시나리오 작성	정황 시나리오는 사용자의 요구사항을 도출하기 위해 작성하는 것으로, 사용자가 목표를 달성하기 위해 수행하는 방법을 순차적으로 묘사한 것이다. • 정황 시나리오는 요구사항 정의에 사용되는 초기 시나리오이다. • 정황 시나리오는 개발하는 서비스의 모습을 상상하는 첫 번째 단계로 사용자 관점에서 시나리오를 작성해야 한다. • 사용자가 주로 사용하는 기능 위주로 작성해야 하며, 함께 발생하는 기능들은 하나의 시나리오에 통합한다.
요구사항 작성	요구사항은 정황 시나리오를 토대로 작성한다.

핵심 028 UI 요구사항 확인

UI 요구사항 확인은 새로 개발할 시스템에 적용할 UI 관련 요구사항을 조사해서 작성하는 단계로, 다양한 경로를 통해 사용자의 요구사항을 조사하고 분석한 후 작성해야 한다.

• UI 요구사항 확인 순서

목표 정의	목표 정의 단계에서는 사용자들을 대상으로 인터뷰를 진행한 후 사용자들의 의견이 수렴된 비즈니스 요구사항을 정의한다. • 인터뷰를 통해 사업적, 기술적인 요구사항을 명확히 이해한다.
활동 사항 정의	활동 사항 정의 단계에서는 조사한 요구사항을 토대로 앞으로 해야 할 활동 사항을 정의한다. • 사용자와 회사의 비전을 일치시키는 작업을 진행한다. • 기술의 발전 가능성을 파악하고 UI 디자인의 방향을 제시한다.
UI 요구사항 작성	UI 요구사항을 작성할 때는 여러 경로를 통해 수집된 사용자들의 요구사항을 검토하고 분석하여 UI 개발 목적에 맞게 작성해야 한다. • UI 요구사항은 반드시 실사용자 중심으로 작성되어야 한다.

핵심 029 품질 요구사항

소프트웨어의 품질은 소프트웨어의 기능, 성능, 만족도 등 소프트웨어에 대한 요구사항이 얼마나 충족하는가를 나타내는 소프트웨어 특성의 총체이다.

- 소프트웨어의 품질은 사용자의 요구사항을 충족시킴으로써 확립된다.
- ISO/IEC 9126 : 소프트웨어의 품질 특성과 평가를 위한 표준 지침으로서 국제 표준으로 널리 사용됨
- ISO/IEC 25010 : 소프트웨어 제품에 대한 국제 표준으로, 2011년에 ISO/IEC 9126을 개정하여 만들었음
- ISO/IEC 12119 : ISO/IEC 9126을 준수한 품질 표준으로, 테스트 절차를 포함하여 규정함
- ISO/IEC 14598 : 소프트웨어 품질의 측정과 평가에 필요 절차를 규정한 표준으로, 개발자, 구매자, 평가자 별로 수행해야 할 제품 평가 활동을 규정함

핵심

030

ISO/IEC 9126의 소프트웨어 품질 특성

기능성 (Functionality)	<ul style="list-style-type: none"> 소프트웨어가 사용자의 요구사항을 정확하게 만족하는 기능을 제공하는지 여부를 나타낸다. 하위 특성 : 적절성/적합성(Suitability), 정밀성/정확성(Accuracy), 상호 운용성(Interoperability), 보안성(Security), 준수성(Compliance)
신뢰성 (Reliability)	<ul style="list-style-type: none"> 소프트웨어가 요구된 기능을 정확하고 일관되게 오류 없이 수행할 수 있는 정도를 나타낸다. 하위 특성 : 성숙성(Maturity), 고장 허용성(Fault Tolerance), 회복성(Recoverability)
사용성 (Usability)	<ul style="list-style-type: none"> 사용자와 컴퓨터 사이에 발생하는 어떠한 행위에 대하여 사용자가 정확하게 이해하고 사용하며, 향후 다시 사용하고 싶은 정도를 나타낸다. 하위 특성 : 이해성(Understandability), 학습성(Learnability), 운용성(Operability), 친밀성(Attractiveness)
효율성 (Efficiency)	<ul style="list-style-type: none"> 사용자가 요구하는 기능을 할당된 시간 동안 한 정된 자원으로 얼마나 빨리 처리할 수 있는지 정도를 나타낸다. 하위 특성 : 시간 효율성(Time Behaviour), 자원 효율성(Resource Behaviour)
유지 보수성 (Maintainability)	<ul style="list-style-type: none"> 환경의 변화 또는 새로운 요구사항이 발생했을 때 소프트웨어를 개선하거나 확장할 수 있는 정도를 나타낸다. 하위 특성 : 분석성(Analyzability), 변경성(Changeability), 안정성(Stability), 시험성(Testability)
이식성 (Portability)	<ul style="list-style-type: none"> 소프트웨어가 다른 환경에서도 얼마나 쉽게 적용할 수 있는지 정도를 나타낸다. 하위 특성 : 적응성(Adaptability), 설치성(Installability), 대체성(Replaceability), 공존성(Co-existence)

핵심

031

UI 프로토타입 제작 및 검토

UI 프로토타입의 개요

- 프로토타입은 사용자 요구사항을 기반으로 실제 동작하는 것처럼 만든 동적인 형태의 모형으로, 테스트가 가능하다.
- 프로토타입은 사용자의 요구사항을 개발자가 맞게 해석했는지 검증하기 위한 것으로, 최대한 간단하게 만들어야 한다.
- 프로토타입은 일부 핵심적인 기능만을 제공하지만 최종 제품의 작동 방식을 이해시키는데 필요한 기능은 반드시 포함되어야 한다.
- 프로토타이핑 및 테스트를 거치지 않고는 실제 사용자와 제품 간의 상호 작용 방식을 예측하기 어려우므로 실제 사용자를 대상으로 테스트하는 것이 좋다.

UI 프로토타입의 장·단점

장점	<ul style="list-style-type: none"> 사용자를 설득하고 이해시키기 쉽다. 요구사항과 기능의 불일치 등으로 인한 혼선을 예방할 수 있어 개발 시간을 줄일 수 있다. 사전에 오류를 발견할 수 있다.
단점	<ul style="list-style-type: none"> 프로토타입에 사용자의 모든 요구사항을 반영하기 위한 반복적인 개선 및 보완 작업 때문에 작업 시간을 증가시킬 수 있고, 필요 이상으로 자원을 소모할 수 있다. 부분적으로 프로토타이핑을 진행하다보면 중요한 작업이 생략될 수 있다.

프로토타이핑의 종류

페이퍼 프로토타입 (Paper Prototype)	<ul style="list-style-type: none"> 아날로그적인 방법으로, 스케치, 그림, 글 등을 이용하여 손으로 직접 작성하는 방법이다. 제작 기간이 짧은 경우, 제작 비용이 적을 경우, 업무 협의가 빠를 경우 사용한다.
디지털 프로토타입 (Digital Prototype)	<ul style="list-style-type: none"> 파워포인트, 아크로벳, 비지오, 옴니그래플 등과 같은 프로그램을 사용하여 작성하는 방법이다. 재사용이 필요한 경우, 산출물과 비슷한 효과가 필요한 경우, 숙련된 전문가가 있을 경우 사용한다.

핵심 032 UI 설계서 작성

UI 설계서의 개요

- UI 설계서는 사용자의 요구사항을 바탕으로 UI 설계를 구체화하여 작성하는 문서로, 상세 설계 전에 대표적인 화면들을 설계한다.
- UI 설계서는 기획자, 개발자, 디자이너 등과의 원활한 의사소통을 위해 작성한다.
- 작성 순서

UI 설계서 표지 작성	UI 설계서 표지는 다른 문서와 혼동되지 않도록 프로젝트명 또는 시스템명을 포함시켜 작성한다.
UI 설계서 개정 이력 작성	<ul style="list-style-type: none"> • UI 설계서 개정 이력은 UI 설계서가 수정될 때마다 어떤 부분이 어떻게 수정되었는지를 정리해 놓은 문서이다. • 처음 작성 시 첫 번째 항목을 '초안 작성', 버전 (Version)을 1.0으로 설정한다. • UI 설계서에 변경 사항이 있을 때마다 변경 내용을 적고 버전을 0.1씩 높인다.
UI 요구사항 정의서 작성	UI 요구사항 정의서는 사용자의 요구사항을 확인하고 정리한 문서로, 사용자 요구사항의 UI 적용 여부를 요구사항별로 표시한다.
시스템 구조 작성	시스템 구조는 UI 요구사항과 UI 프로토타입에 기초하여 전체 시스템의 구조를 설계한 것으로 사용자의 요구사항이 어떻게 시스템에 적용되는지 알 수 있다.
사이트 맵 (Site Map) 작성	<ul style="list-style-type: none"> • 사이트 맵은 시스템 구조를 바탕으로 사이트에 표시할 콘텐츠를 한 눈에 알아 볼 수 있도록 메뉴 별로 구분하여 설계한 것이다. • 사이트 맵을 작성한 후 사이트 맵의 상세 내용 (Site Map Detail)을 표 형태로 작성한다.
프로세스 (Process) 정의서 작성	프로세스 정의서는 사용자 관점에서 사용자가 요구하는 프로세스들을 작업 진행 순서에 맞춰 정리한 것으로 UI의 전체적인 흐름을 파악할 수 있다.
화면 설계	<ul style="list-style-type: none"> • 화면 설계는 UI 프로토타입과 UI 프로세스를 참고하여 필요한 화면을 페이지별로 설계한 것이다. • 화면을 구분하기 위해 화면별 고유ID를 부여하고 별도 표지를 작성한다.

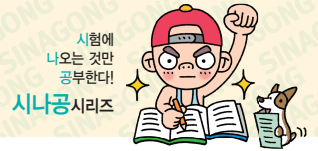
핵심 033 UI 시나리오 문서

UI 시나리오 문서 개요

- UI 상세 설계는 UI 설계서를 바탕으로 실제 설계 및 구현을 위해 모든 화면에 대한 자세한 설계를 진행하는 단계로, UI 상세 설계를 할 때는 반드시 시나리오를 작성해야 한다.
- UI 시나리오 문서는 사용자 인터페이스의 기능 구조, 대표 화면, 화면 간 인터랙션의 흐름, 다양한 상황에서의 예외 처리 등을 문서로 정리한 것이다.
- UI 설계자 또는 인터랙션 디자이너가 UI 시나리오 문서를 작성하면 그래픽 디자이너가 시나리오를 바탕으로 디자인을 하고 개발자가 UI를 구현한다.

UI 시나리오 문서의 요건

완전성 (Complete)	<ul style="list-style-type: none"> • 누락되지 않도록 최대한 상세하게 기술해야 한다. • 해당 시스템의 기능보다는 사용자의 태스크에 초점을 맞춰 기술한다.
일관성 (Consistent)	서비스 목표, 시스템 및 사용자의 요구사항, UI 스타일 등이 모두 일관성을 유지해야 한다.
이해성 (Understandable)	<ul style="list-style-type: none"> • 누구나 쉽게 이해할 수 있도록 설명한다. • 불분명하거나 추상적인 표현은 피한다.
가독성 (Readable)	<ul style="list-style-type: none"> • 표준화된 템플릿 등을 활용하여 문서를 쉽게 읽을 수 있도록 해야 한다. • v1.0, v2.0 등과 같이 문서 인덱스에 대한 규칙이나 목차를 제공한다. • 읽기 쉽도록 줄 간격, 단락, 들여쓰기 등의 기준을 마련한다. • 시각적인 효과를 위해 여백이나 빈 페이지, 하이라이팅을 일관성 있게 지정한다. • 하이퍼링크 등을 지정하여 문서들이 서로 참조될 수 있도록 지정한다.
수정 용이성 (Modifiable)	시나리오의 수정이나 개선이 쉬워야 한다.
추적 용이성 (Traceable)	변경 사항은 언제, 어떤 부분이, 왜 발생했는지 쉽게 추적할 수 있어야 한다.



핵심 034 HCI / UX / 감성공학

HCI(Human Computer Interaction or Interface)

- HCI는 사람이 시스템을 보다 편리하고 안전하게 사용할 수 있도록 연구하고 개발하는 학문으로, 최종 목표는 시스템을 사용하는데 있어 최적의 사용자 경험(UX)을 만드는 것이다.
- HCI는 어떤 제품이 좋은 제품인지, 어떻게 하면 좋은 제품을 만들 수 있는지 등을 연구한다.

UX(User Experience)

- UX는 사용자가 시스템이나 서비스를 이용하면서 느끼고 생각하게 되는 총체적인 경험을 말한다. 단순히 기능이나 절차상의 만족뿐만 아니라 사용자가 참여, 사용, 관찰하고, 상호 교감을 통해서 알 수 있는 가치 있는 경험을 말한다.
- UX는 기술을 효용성 측면에서만 보는 것이 아니라 사용자의 삶의 질을 향상시키는 하나의 방향으로 보는 새로운 개념이다.
- UI가 사용성, 접근성, 편의성을 중시한다면 UX는 이러한 UI를 통해 사용자가 느끼는 만족이나 감정을 중시한다.
- UX의 특징
 - 주관성(Subjectivity) : 사람들의 개인적, 신체적, 인지적 특성에 따라 다르므로 주관적이다.
 - 정황성(Contextuality) : 경험이 일어나는 상황 또는 주변 환경에 영향을 받는다.
 - 총체성(Holistic) : 개인이 느끼는 총체적인 심리적, 감성적인 결과이다.

감성공학

- 감성공학은 제품이나 작업환경을 사용자의 감성에 알맞도록 설계 및 제작하는 기술로, 인문사회과학, 공학, 의학 등 여러 분야의 학문이 공존하는 종합과학이다.
- ‘감성’을 과학적으로 측정하기 위해서는 생체계측 기술, 감각계측 기술, 센서, 인공지능, 생체제어 기술 등이 요구된다.
- 감성공학의 목적은 인간의 삶을 편리하고 안전하며 쾌적하게 만드는 것이다.

핵심 035 소프트웨어 아키텍처의 설계

소프트웨어 아키텍처는 소프트웨어의 골격이 되는 기본 구조이자, 소프트웨어를 구성하는 요소들 간의 관계를 표현하는 시스템의 구조 또는 구조체이다.

- 소프트웨어 개발 시 적용되는 원칙과 지침이며, 이해관계자들의 의사소통 도구로 활용된다.
- 소프트웨어 아키텍처 설계의 기본 원리로는 모듈화, 추상화, 단계적 분해, 정보은닉이 있다.

※ 상위 설계와 하위 설계

소프트웨어 개발의 설계 단계는 크게 상위 설계와 하위 설계로 구분할 수 있다.

	상위 설계	하위 설계
별칭	아키텍처 설계, 예비 설계	모듈 설계, 상세 설계
설계 대상	시스템의 전체적인 구조	시스템의 내부 구조 및 행위
세부 목록	구조, DB, 인터페이스	컴포넌트, 자료 구조, 알고리즘

핵심 036 소프트웨어 아키텍처 설계의 기본 원리

모듈화 (Modularity)	모듈화란 소프트웨어의 성능을 향상시키거나 시스템의 수정 및 재사용, 유지 관리 등이 용이하도록 시스템의 기능들을 모듈 단위로 나누는 것을 의미한다.
추상화 (Abstraction)	추상화는 문제의 전체적이고 포괄적인 개념을 설계한 후 차례로 세분화하여 구체화시켜 나가는 것이다.
단계적 분해 (Stepwise Refinement)	단계적 분해는 Niklaus Wirth에 의해 제안된 하향식 설계 전략으로, 문제를 상위의 중요 개념으로부터 하위의 개념으로 구체화시키는 분할 기법이다.
정보 은닉 (Information Hiding)	정보 은닉은 한 모듈 내부에 포함된 절차와 자료들의 정보가 감추어져 다른 모듈이 접근하거나 변경하지 못하도록 하는 기법이다.

핵심 037 협약(Contract)에 의한 설계

컴포넌트를 설계할 때 클래스에 대한 여러 가정을 공유할 수 있도록 명세한 것으로, 소프트웨어 컴포넌트에 대한 정확한 인터페이스를 명세한다.

- 협약에 의한 설계 시 명세에 포함될 조건에는 선행 조건, 결과 조건, 불변 조건이 있다.

선행 조건 (Precondition)	오퍼레이션이 호출되기 전에 참이 되어야 할 조건
결과 조건 (Postcondition)	오퍼레이션이 수행된 후 만족되어야 할 조건
불변 조건 (Invariant)	오퍼레이션이 실행되는 동안 항상 만족되어야 할 조건

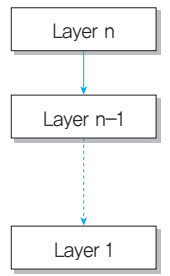
핵심 038 아키텍처 패턴(Patterns)의 개요

아키텍처 패턴은 아키텍처를 설계할 때 참조할 수 있는 전형적인 해결 방식 또는 예제를 의미한다.

- 아키텍처 패턴은 소프트웨어 시스템의 구조를 구성하기 위한 기본적인 윤곽을 제시한다.
- 아키텍처 패턴에는 서브시스템들과 그 역할이 정의되어 있으며, 서브시스템 사이의 관계와 여러 규칙·지침 등이 포함되어 있다.
- 아키텍처 패턴을 아키텍처 스타일 또는 표준 아키텍처라고도 한다.
- 아키텍처 패턴의 장점
 - 시행착오를 줄여 개발 시간을 단축시키고, 고품질의 소프트웨어를 생산할 수 있다.
 - 검증된 구조로 개발하기 때문에 안정적인 개발이 가능하다.
 - 이해관계자들이 공통된 아키텍처를 공유할 수 있어 의사소통이 간편해진다.
 - 시스템의 구조를 이해하는 것이 쉬워 개발에 참여하지 않은 사람도 손쉽게 유지보수를 수행할 수 있다.
 - 시스템의 특성을 개발 전에 예측하는 것이 가능해진다.

핵심 039 레이어 패턴(Layers pattern)

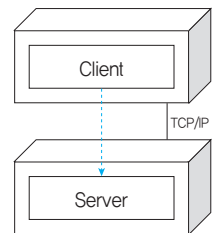
레이어 패턴은 시스템을 계층(Layer)으로 구분하여 구성하는 고전적인 방법 중의 하나다.



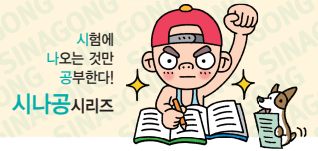
- 레이어 패턴은 각각의 서브시스템들이 계층 구조를 이루며, 상위 계층은 하위 계층에 대한 서비스 제공자가 되고, 하위 계층은 상위 계층의 클라이언트가 된다.
- 레이어 패턴은 서로 마주보는 두 개의 계층 사이에서만 상호작용이 이루어지며, 변경 사항을 적용할 때도 서로 마주보는 두 개의 계층에만 영향을 미치므로 변경 작업이 용이하다.
- 레이어 패턴은 특정 계층만을 교체해 시스템을 개선하는 것이 가능하다.
- 대표적으로 OSI 참조 모델이 있다.

핵심 040 클라이언트-서버 패턴(Client-Server Pattern)

클라이언트-서버 패턴은 하나의 서버 컴포넌트와 다수의 클라이언트 컴포넌트로 구성되는 패턴이다.



- 클라이언트-서버 패턴에서 사용자는 클라이언트와만 의사소통을 한다. 즉 사용자가 클라이언트를 통해 서버에 요청하고 클라이언트가 응답을 받아 사용자에게 제공하는 방식으로 서비스를 제공한다.
- 서버는 클라이언트의 요청에 대비해 항상 대기 상태를 유지해야 한다.
- 클라이언트나 서버는 요청과 응답을 받기 위해 동기화되는 경우를 제외하고는 서로 독립적이다.



핵심 041 파이프-필터 패턴 (Pipe-Filter Pattern)

파이프-필터 패턴은 데이터 스트림 절차의 각 단계를 필터(Filter) 컴포넌트로 캡슐화하여 파이프(Pipe)를 통해 데이터를 전송하는 패턴이다.

- 필터 컴포넌트는 재사용성이 좋고, 추가가 쉬워 확장이 용이하다.
- 필터 컴포넌트들을 재배치하여 다양한 파이프라인을 구축하는 것이 가능하다.
- 파이프-필터 패턴은 데이터 변환, 버퍼링, 동기화 등에 주로 사용된다.
- 대표적으로 UNIX의 셸(Shell)이 있다.

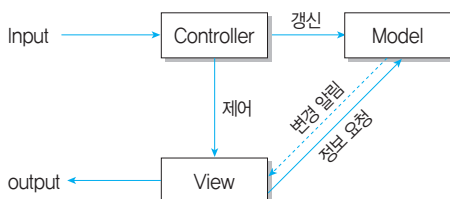


핵심 042 모델-뷰-컨트롤러 패턴(Model-View-Controller Pattern)

모델-뷰-컨트롤러 패턴은 서브시스템을 3개의 부분으로 구조화하는 패턴이며, 각 부분의 역할은 다음과 같다.

모델(Model)	서브시스템의 핵심 기능과 데이터를 보관한다.
뷰(View)	사용자에게 정보를 표시한다.
컨트롤러(Controller)	사용자로부터 받은 입력을 처리한다.

- 모델-뷰-컨트롤러 패턴의 각 부분은 별도의 컴포넌트로 분리되어 있으므로 서로 영향을 받지 않고 개발 작업을 수행할 수 있다.
- 모델-뷰-컨트롤러 패턴에서는 여러 개의 뷰를 만들 수 있으므로 한 개의 모델에 대해 여러 개의 뷰를 필요로 하는 대화형 애플리케이션에 적합하다.



핵심 043 기타 패턴

마스터-슬레이브 패턴 (Master-Slave Pattern)	<ul style="list-style-type: none"> • 마스터 컴포넌트에서 슬레이브 컴포넌트로 작업을 분할한 후, 슬레이브 컴포넌트에서 처리된 결과물을 다시 돌려받는 방식으로 작업을 수행하는 패턴이다. • 장애 허용 시스템과 병렬 컴퓨팅 시스템에서 주로 활용된다.
브로커 패턴 (Broker Pattern)	<ul style="list-style-type: none"> • 사용자가 원하는 서비스와 특성을 브로커 컴포넌트에 요청하면 브로커 컴포넌트가 요청에 맞는 컴포넌트와 사용자를 연결해 준다. • 분산 환경 시스템에서 주로 활용된다.
피어-투-피어 패턴 (Peer-To-Peer Pattern)	피어(Peer)를 하나의 컴포넌트로 간주하며, 각 피어는 서비스를 호출하는 클라이언트가 될 수도, 서비스를 제공하는 서버가 될 수도 있는 패턴이다.
이벤트-버스 패턴 (Event-Bus Pattern)	소스가 특정 채널에 이벤트 메시지를 발행(Publish)하면, 해당 채널을 구독(Subscribe)한 리스너들이 메시지를 받아 이벤트를 처리하는 방식이다.
블랙보드 패턴 (Blackboard Pattern)	<ul style="list-style-type: none"> • 모든 컴포넌트들이 공유 데이터 저장소와 블랙보드 컴포넌트에 접근이 가능한 형태로, 컴포넌트들은 검색을 통해 블랙보드에서 원하는 데이터를 찾을 수 있다. • 음성 인식, 차량 식별, 신호 해석 등에 주로 활용된다.
인터프리터 패턴 (Interpreter Pattern)	프로그램 코드의 각 라인을 수행하는 방법을 지정하고, 기호마다 클래스를 갖도록 구성된다.

핵심 044 객체(Object)

객체는 데이터와 데이터를 처리하는 함수를 묶어 놓은 (캡슐화한) 하나의 소프트웨어 모듈이다.

데이터	<ul style="list-style-type: none"> • 객체가 가지고 있는 정보로 속성이나 상태, 분류 등을 나타낸다. • 속성(Attribute), 상태, 변수, 상수, 자료 구조라고도 한다.
함수	<ul style="list-style-type: none"> • 객체가 수행하는 기능으로 객체가 갖는 데이터(속성, 상태)를 처리하는 알고리즘이다. • 객체의 상태를 참조하거나 변경하는 수단이 되는 것으로 메소드(Method, 행위), 서비스(Service), 동작(Operation), 연산이라고도 한다.

• 객체의 특성

- 객체는 독립적으로 식별 가능한 이름을 가지고 있다.
- 객체가 가질 수 있는 조건을 상태(State)라고 하는데, 일반적으로 상태는 시간에 따라 변한다.
- 객체와 객체는 상호 연관성에 의한 관계가 형성된다.
- 객체가 반응할 수 있는 메시지(Message)의 집합을 행위라고 하며, 객체는 행위의 특징을 나타낼 수 있다.
- 객체는 일정한 기억장소를 가지고 있다.
- 객체의 메소드는 다른 객체로부터 메시지를 받았을 때 정해진 기능을 수행한다.

핵심 045 클래스(Class)

클래스는 공통된 속성과 연산(행위)을 갖는 객체의 집합으로, 객체의 일반적인 타입(Type)을 의미한다.

- 클래스는 각각의 객체들이 갖는 속성과 연산을 정의하고 있는 틀이다.
- 클래스는 객체지향 프로그램에서 데이터를 추상화하는 단위이다.
- 클래스에 속한 각각의 객체를 인스턴스(Instance)라 하며, 클래스로부터 새로운 객체를 생성하는 것을 인스턴스화(Instantiation)라고 한다.

핵심 046 캡슐화(Encapsulation)

캡슐화는 데이터(속성)와 데이터를 처리하는 함수를 하나로 묶는 것을 의미한다.

- 캡슐화된 객체는 인터페이스를 제외한 세부 내용이 은폐(정보 은닉)되어 외부에서의 접근이 제한적이기 때문에 외부 모듈의 변경으로 인한 파급 효과가 적다.
- 캡슐화된 객체들은 재사용이 용이하다.
- 객체들 간의 메시지를 주고받을 때 상대 객체의 세부 내용은 알 필요가 없으므로 인터페이스가 단순해지고, 객체 간의 결합도가 낮아진다.

핵심 047 상속(Inheritance)

상속은 이미 정의된 상위 클래스(부모 클래스)의 모든 속성과 연산을 하위 클래스(자식 클래스)가 물려받는 것이다.

- 상속을 이용하면 하위 클래스는 상위 클래스의 모든 속성과 연산을 자신의 클래스 내에서 다시 정의하지 않고서도 즉시 자신의 속성으로 사용할 수 있다.
- 하위 클래스는 상위 클래스로부터 상속받은 속성과 연산 외에 새로운 속성과 연산을 첨가하여 사용할 수 있다.

핵심 048 연관성(Relationship)

연관성은 두 개 이상의 객체(클래스)들이 상호 참조하는 관계를 말하며 종류는 다음과 같다.

종류	의미	특징
is member of	연관화 (Association)	2개 이상의 객체가 상호 관련되어 있음을 의미함
is instance of	분류화 (Classification)	동일한 형의 특성을 갖는 객체들을 모아 구성하는 것
is part of	집단화 (Aggregation)	관련 있는 객체들을 묶어 하나의 상위 객체를 구성하는 것
is a	일반화 (Generalization)	공통적인 성질들로 추상화한 상위 객체를 구성하는 것
	특수화/상세화 (Specialization)	상위 객체를 구체화하여 하위 객체를 구성하는 것

핵심 049 객체지향 분석의 방법론

객체지향 분석을 위한 여러 방법론이 제시되었으며 각 방법론은 다음과 같다.

- Rumbaugh(럼바우) 방법 : 가장 일반적으로 사용되는 방법으로 분석 활동을 객체 모델, 동적 모델, 기능 모델로 나누어 수행하는 방법이다.

- Booch(부치) 방법 : 미시적(Micro) 개발 프로세스와 거시적(Macro) 개발 프로세스를 모두 사용하는 분석 방법으로, 클래스와 객체들을 분석 및 식별하고 클래스의 속성과 연산을 정의한다.
- Jacobson 방법 : Use Case를 강조하여 사용하는 분석 방법이다.
- Coad와 Yourdon 방법 : E-R 다이어그램을 사용하여 객체의 행위를 모델링하며, 객체 식별, 구조 식별, 주제 정의, 속성과 인스턴스 연결 정의, 연산과 메시지 연결 정의 등의 과정으로 구성하는 기법이다.
- Wirfs-Brock 방법 : 분석과 설계 간의 구분이 없고, 고객 명세서를 평가해서 설계 작업까지 연속적으로 수행하는 기법이다.

핵심 050 럼바우(Rumbaugh)의 분석 기법

럼바우의 분석 기법은 모든 소프트웨어 구성 요소를 그래픽 표기법을 이용하여 모델링하는 기법으로, 객체 모델링 기법(OMT, Object-Modeling Technique)이라고도 한다.

- 분석 활동은 '객체 모델링 → 동적 모델링 → 기능 모델링' 순으로 통해 이루어진다.

객체 모델링 (Object Modeling)	정보 모델링이라고도 하며, 시스템에서 요구되는 객체를 찾아내어 속성과 연산 식별 및 객체들 간의 관계를 규정하여 객체 다이어그램으로 표시하는 것이다.
동적 모델링 (Dynamic Modeling)	상태 다이어그램(상태도)을 이용하여 시간의 흐름에 따른 객체들 간의 제어 흐름, 상호 작용, 동작 순서 등의 동적인 행위를 표현하는 모델링이다.
기능 모델링 (Functional Modeling)	자료 흐름도(DFD)를 이용하여 다수의 프로세스들 간의 자료 흐름을 중심으로 처리 과정을 표현한 모델링이다.

핵심 051 객체지향 설계 원칙

객체지향 설계 원칙은 시스템 변경이나 확장에 유연한 시스템을 설계하기 위해 지켜야 할 다섯 가지 원칙으로, 다섯 가지 원칙의 앞 글자를 따 SOLID 원칙이라고도 불린다.

단일 책임 원칙 (SRP, Single Responsibility Principle)	객체는 단 하나의 책임만 가져야 한다는 원칙이다.
개방-폐쇄 원칙 (OCP, Open-Closed Principle)	기존의 코드를 변경하지 않고 기능을 추가할 수 있도록 설계해야 한다는 원칙이다.
리스코프 치환 원칙 (LSP, Liskov Substitution Principle)	자식 클래스는 최소한 자신의 부모 클래스에서 가능한 행위는 수행할 수 있어야 한다는 설계 원칙이다.
인터페이스 분리 원칙 (ISP, Interface Segregation Principle)	자신이 사용하지 않는 인터페이스와 의존 관계를 맺거나 영향을 받지 않아야 한다는 원칙이다.
의존 역전 원칙 (DIP, Dependency Inversion Principle)	각 객체들 간의 의존 관계가 성립될 때, 추상성이 낮은 클래스보다 추상성이 높은 클래스와 의존 관계를 맺어야 한다는 원칙이다.

핵심 052 결합도(Coupling)

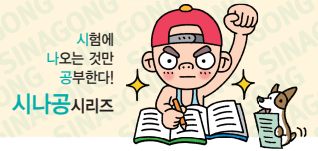
결합도는 모듈 간에 상호 의존하는 정도 또는 두 모듈 사이의 연관 관계를 의미한다.

- 다양한 결합으로 모듈을 구성할 수 있으나 결합도가 약할수록 품질이 높고, 강할수록 품질이 낮다.
- 결합도가 강하면 시스템 구현 및 유지보수 작업이 어렵다.
- 결합도의 종류에는 자료 결합도, 스탬프 결합도, 제어 결합도, 외부 결합도, 공통 결합도, 내용 결합도가 있으며 결합도의 정도는 다음과 같다.

자료 결합도	스탬프 결합도	제어 결합도	외부 결합도	공통 결합도	내용 결합도
--------	---------	--------	--------	--------	--------

결합도 약함 ←

→ 결합도 강함



자료 결합도 (Data Coupling)	모듈 간의 인터페이스가 자료 요소로만 구성 될 때의 결합도
스탬프(검인) 결합도 (Stamp Coupling)	모듈 간의 인터페이스로 배열이나 레코드 등 의 자료 구조가 전달될 때의 결합도
제어 결합도 (Control Coupling)	어떤 모듈이 다른 모듈 내부의 논리적인 흐름을 제어하기 위해 제어 신호를 이용하여 통신하거나 제어 요소(Function Code, Switch, Tag, Flag)를 전달하는 결합도
외부 결합도 (External Coupling)	어떤 모듈에서 선언한 데이터(변수)를 외부의 다른 모듈에서 참조할 때의 결합도
공통(공유) 결합도 (Common Coupling)	공유되는 공통 데이터 영역을 여러 모듈이 사용할 때의 결합도
내용 결합도 (Content Coupling)	한 모듈이 다른 모듈의 내부 기능 및 그 내부 자료를 직접 참조하거나 수정할 때의 결합도

교환(통신)적 응집도 (Communication Cohesion)	동일한 입력과 출력을 사용하여 서로 다른 기능을 수행하는 구성 요소들이 모였을 경우의 응집도
절차적 응집도 (Procedural Cohesion)	모듈이 다수의 관련 기능을 가질 때 모듈 안의 구성 요소들이 그 기능을 순차적으로 수행할 경우의 응집도
시간적 응집도 (Temporal Cohesion)	특정 시간에 처리되는 몇 개의 기능을 모아 하나의 모듈로 작성할 경우의 응집도
논리적 응집도 (Logical Cohesion)	유사한 성격을 갖거나 특정 형태로 분류되는 처리 요소들로 하나의 모듈이 형성되는 경우의 응집도
우연적 응집도 (Coincidental Cohesion)	모듈 내부의 각 구성 요소들이 서로 관련 없는 요소로만 구성된 경우의 응집도

핵심 053 응집도(Cohesion)

응집도는 정보 은닉 개념을 확장한 것으로, 명령어나 호출문 등 모듈의 내부 요소들의 서로 관련되어 있는 정도, 즉 모듈이 독립적인 기능으로 정의되어 있는 정도를 의미한다.

- 다양한 기준으로 모듈을 구성할 수 있으나 응집도가 강할수록 품질이 높고, 약할수록 품질이 낮다.
- 응집도의 종류에는 기능적 응집도, 순차적 응집도, 교환(통신)적 응집도, 절차적 응집도, 시간적 응집도, 논리적 응집도, 우연적 응집도가 있으며 응집도의 정도는 다음과 같다.

기능적 응집도	순차적 응집도	교환적 응집도	절차적 응집도	시간적 응집도	논리적 응집도	우연적 응집도
---------	---------	---------	---------	---------	---------	---------

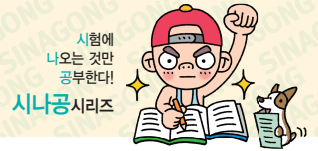
응집도 강함 ← → 응집도 약함

기능적 응집도 (Functional Cohesion)	모듈 내부의 모든 기능 요소들이 단일 문제와 연관되어 수행될 경우의 응집도
순차적 응집도 (Sequential Cohesion)	모듈 내 하나의 활동으로부터 나온 출력 데이터를 그 다음 활동의 입력 데이터로 사용할 경우의 응집도

핵심 054 N-S 차트 (Nassi-Schneiderman Chart)

N-S 차트는 논리의 기술에 중점을 둔 도형을 이용한 표현 방법으로 박스 다이어그램, Chapin Chart라고도 한다.

- 연속, 선택 및 다중 선택, 반복 등의 제어 논리 구조를 표현한다.
- GOTO나 화살표를 사용하지 않는다.
- 조건이 복합되어 있는 곳의 처리를 시각적으로 명확히 식별하는 데 적합하다.
- 선택과 반복 구조를 시각적으로 표현한다.
- 이해하기 쉽고, 코드 변환이 용이하다.
- 읽기는 쉽지만 작성하기가 어려우며, 임의로 제어를 전이하는 것이 불가능하다.
- 총체적인 구조 표현과 인터페이스를 나타내기가 어렵다.
- 단일 입구와 단일 출구로 표현한다.



핵심 055 공통 모듈의 개요

공통 모듈은 여러 프로그램에서 공통적으로 사용할 수 있는 모듈을 의미한다.

- 자주 사용되는 계산식이나 매번 필요한 사용자 인증과 같은 기능들이 공통 모듈로 구성될 수 있다.
- 모듈의 재사용성 확보와 중복 개발 회피를 위해 설계 과정에서 공통 부분을 식별하고 명세를 작성할 필요가 있다.
- 공통 모듈을 구현할 때는 다른 개발자들이 해당 기능을 명확히 이해할 수 있도록 다음의 명세 기법을 준수해야 한다.

정확성 (Correctness)	시스템 구현 시 해당 기능이 필요하다는 것을 알 수 있도록 정확히 작성한다.
명확성 (Clarity)	해당 기능을 이해할 때 중의적으로 해석되지 않도록 명확하게 작성한다.
완전성 (Completeness)	시스템 구현을 위해 필요한 모든 것을 기술한다.
일관성 (Consistency)	공통 기능들 간 상호 충돌이 발생하지 않도록 작성한다.
추적성 (Traceability)	기능에 대한 요구사항의 출처, 관련 시스템 등의 관계를 파악할 수 있도록 작성한다.

핵심 056 재사용(Reuse)

재사용은 비용과 개발 시간을 절약하기 위해 이미 개발된 기능들을 파악하고 재구성하여 새로운 시스템 또는 기능 개발에 사용하기 적합하도록 최적화 시키는 작업이다.

- 재사용을 위해서는 누구나 이해할 수 있고 사용이 가능하도록 사용법을 공개해야 한다.
- 재사용되는 대상은 외부 모듈과의 결합도는 낮고, 응집도는 높아야 한다.
- 재사용 규모에 따른 분류

함수와 객체	클래스나 메소드 단위의 소스 코드를 재사용한다.
컴포넌트	컴포넌트 자체에 대한 수정 없이 인터페이스를 통해 통신하는 방식으로 재사용한다.
애플리케이션	공통된 기능들을 제공하는 애플리케이션을 공유하는 방식으로 재사용한다.

핵심 057 효과적인 모듈 설계 방안

- 결합도는 줄이고 응집도는 높여서 모듈의 독립성과 재사용성을 높인다.
- 모듈의 제어 영역 안에서 그 모듈의 영향 영역을 유지시킨다.
- 복잡도와 중복성을 줄이고 일관성을 유지시킨다.
- 모듈의 기능은 예측이 가능해야 하며 지나치게 제한적이어서는 안 된다.
- 유지보수가 용이해야 한다.
- 모듈 크기는 시스템의 전반적인 기능과 구조를 이해하기 쉬운 크기로 분해한다.
- 효과적인 제어를 위해 모듈 간의 계층적 관계를 정의하는 자료가 제시되어야 한다.

핵심 058 코드(Code)의 개요

코드는 컴퓨터를 이용하여 자료를 처리하는 과정에서 분류·조합 및 집계를 용이하게 하고, 특정 자료의 추출을 쉽게 하기 위해서 사용하는 기호이다.

- 코드는 정보를 신속·정확·명료하게 전달할 수 있게 한다.
- 일반적인 코드의 예로 주민등록번호, 학번, 전화번호 등이 있다.
- 코드의 주요 기능에는 식별 기능, 분류 기능, 배열 기능, 표준화 기능, 간소화 기능이 있다.

식별 기능	데이터 간의 성격에 따라 구분이 가능하다.
분류 기능	특정 기준이나 동일한 유형에 해당하는 데이터를 그룹화 할 수 있다.
배열 기능	의미를 부여하여 나열할 수 있다.
표준화 기능	다양한 데이터를 기준에 맞추어 표현할 수 있다.
간소화 기능	복잡한 데이터를 간소화할 수 있다.

핵심 059 코드의 종류

순차 코드 (Sequence Code)	자료의 발생 순서, 크기 순서 등 일정 기준에 따라서 최초의 자료부터 차례로 일련번호를 부여하는 방법으로, 순서 코드 또는 일련번호 코드라고도 한다. 예 1, 2, 3, 4, ...
블록 코드 (Block Code)	코드화 대상 항목 중에서 공통성이 있는 것끼리 블록으로 구분하고, 각 블록 내에서 일련번호를 부여하는 방법으로, 구분 코드라고도 한다. 예 1001~1100 : 총무부, 1101~1200 : 영업부
10진 코드 (Decimal Code)	코드화 대상 항목을 0~9까지 10진 분할하고, 다시 그 각각에 대하여 10진 분할하는 방법을 필요한 만큼 반복하는 방법으로, 도서 분류식 코드라고도 한다. 예 1000 : 공학, 1100 : 소프트웨어 공학, 1110 : 소프트웨어 설계
그룹 분류 코드 (Group Classification Code)	코드화 대상 항목을 일정 기준에 따라 대분류, 중분류, 소분류 등으로 구분하고, 각 그룹 안에서 일련번호를 부여하는 방법이다. 예 1-01-001 : 본사-총무부-인사계, 2-01-001 : 지사-총무부-인사계
연상 코드 (Mnemonic Code)	코드화 대상 항목의 명칭이나 약호와 관계있는 숫자나 문자, 기호를 이용하여 코드를 부여하는 방법이다. 예 TV-40 : 40인치 TV, L-15-220 : 15W 220V의 램프
표의 숫자 코드 (Significant Digit Code)	코드화 대상 항목의 성질, 즉 길이, 넓이, 부피, 지름, 높이 등의 물리적 수치를 그대로 코드에 적용시키는 방법으로, 유효 숫자 코드라고도 한다. 예 120-720-1500 : 두께×폭×길이 120×720×1500인 강판
합성 코드 (Combined Code)	필요한 기능을 하나의 코드로 수행하기 어려운 경우 2개 이상의 코드를 조합하여 만드는 방법이다. 예 연상 코드 + 순차 코드 KE-711 : 대한항공 711기, AC-253 : 에어캐나다 253기

핵심 060 디자인 패턴(Design Pattern)의 개요

디자인 패턴은 각 모듈의 세분화된 역할이나 모듈들 간의 인터페이스와 같은 코드를 작성하는 수준의 세부적인 구현 방안을 설계할 때 참조할 수 있는 전형적인 해결 방식 또는 예제를 의미한다.

- 디자인 패턴은 문제 및 배경, 실제 적용된 사례, 재사용이 가능한 샘플 코드 등으로 구성되어 있다.
- ‘바퀴를 다시 발명하지 마라(Don't reinvent the wheel)’라는 말과 같이, 개발 과정 중에 문제가 발생하면 새로 해결책을 구상하는 것보다 문제에 해당하는 디자인 패턴을 참고하여 적용하는 것이 더 효율적이다.
- GoF의 디자인 패턴은 유형에 따라 생성 패턴 5개, 구조 패턴 7개, 행위 패턴 11개 총 23개의 패턴으로 구성된다.

핵심 061 디자인 패턴 사용의 장·단점

- 범용적인 코딩 스타일로 인해 구조 파악이 용이하다.
- 객체지향 설계 및 구현의 생산성을 높이는 데 적합하다.
- 검증된 구조의 재사용을 통해 개발 시간과 비용이 절약된다.
- 초기 투자 비용이 부담될 수 있다.
- 개발자 간의 원활한 의사소통이 가능하다.
- 설계 변경 요청에 대한 유연한 대처가 가능하다.
- 객체지향을 기반으로 한 설계와 구현을 다루므로 다른 기반의 애플리케이션 개발에는 적합하지 않다.

핵심 062 생성/구조/행위 패턴

생성 패턴(Creational Pattern)

생성 패턴은 객체의 생성과 관련된 패턴으로 총 5개의 패턴이 있다.

- 생성 패턴은 객체의 생성과 참조 과정을 캡슐화 하여 객체가 생성되거나 변경되어도 프로그램의 구조에 영향을 크게 받지 않도록 하여 프로그램에 유연성을 더 해준다.

추상 팩토리 (Abstract Factory)	<ul style="list-style-type: none"> • 구체적인 클래스에 의존하지 않고, 인터페이스를 통해 서로 연관 · 의존하는 객체들의 그룹으로 생성하여 추상적으로 표현한다. • 연관된 서브 클래스를 묶어 한 번에 교체하는 것이 가능하다.
빌더(Builder)	<ul style="list-style-type: none"> • 작게 분리된 인스턴스를 건축 하듯이 조합하여 객체를 생성한다. • 객체의 생성 과정과 표현 방법을 분리하고 있어, 동일한 객체 생성에서도 서로 다른 결과를 만들어 낼 수 있다.
팩토리 메소드 (Factory Method)	<ul style="list-style-type: none"> • 객체 생성을 서브 클래스에서 처리하도록 분리하여 캡슐화한 패턴이다. • 상위 클래스에서 인터페이스만 정의하고 실제 생성은 서브 클래스가 담당한다. • 가상 생성자(Virtual Constructor) 패턴이라고도 한다.
프로토타입 (Prototype)	<ul style="list-style-type: none"> • 원본 객체를 복제하는 방법으로 객체를 생성하는 패턴이다. • 일반적인 방법으로 객체를 생성하며, 비용이 큰 경우 주로 이용한다.
싱글톤 (Singleton)	<ul style="list-style-type: none"> • 하나의 객체를 생성하면 생성된 객체를 어디서든 참조할 수 있지만, 여러 프로세스가 동시에 참조할 수는 없다. • 클래스 내에서 인스턴스가 하나뿐임을 보장하며, 불필요한 메모리 낭비를 최소화 할 수 있다.

구조 패턴(Structural Pattern)

구조 패턴은 클래스나 객체들을 조합하여 더 큰 구조로 만들 수 있게 해주는 패턴으로 총 7개의 패턴이 있다.

- 구조 패턴은 구조가 복잡한 시스템을 개발하기 쉽게 도와준다.

어댑터 (Adapter)	<ul style="list-style-type: none"> • 호환성이 없는 클래스들의 인터페이스를 다른 클래스가 이용할 수 있도록 변환해주는 패턴이다. • 기존의 클래스를 이용하고 싶지만 인터페이스가 일치하지 않을 때 이용한다.
브리지 (Bridge)	<ul style="list-style-type: none"> • 구현부에서 추상층을 분리하여, 서로가 독립적으로 확장할 수 있도록 구성한 패턴이다. • 기능과 구현을 두 개의 별도 클래스로 구현한다.

컴포지트 (Composite)	<ul style="list-style-type: none"> • 여러 객체를 가진 복합 객체와 단일 객체를 구분 없이 다루고자 할 때 사용하는 패턴이다. • 객체들을 트리 구조로 구성하여 디렉터리 안에 디렉터리가 있듯이 복합 객체 안에 복합 객체가 포함되는 구조를 구현할 수 있다.
데코레이터 (Decorator)	<ul style="list-style-type: none"> • 객체 간의 결합을 통해 능동적으로 기능들을 확장할 수 있는 패턴이다. • 임의의 객체에 부가적인 기능을 추가하기 위해 다른 객체들을 덧붙이는 방식으로 구현한다.
퍼사드 (Facade)	<ul style="list-style-type: none"> • 복잡한 서브 클래스들을 피해 더 상위에 인터페이스를 구성함으로써 서브 클래스들의 기능을 간편하게 사용할 수 있도록 하는 패턴이다. • 서브 클래스들 사이의 통합 인터페이스를 제공하는 Wrapper 객체가 필요하다.
플라이웨이트 (Flyweight)	<ul style="list-style-type: none"> • 인스턴스가 필요할 때마다 매번 생성하는 것이 아니고 가능한 한 공유해서 사용함으로써 메모리를 절약하는 패턴이다. • 다수의 유사 객체를 생성하거나 조작할 때 유용하게 사용할 수 있다.
프록시 (Proxy)	<ul style="list-style-type: none"> • 접근이 어려운 객체와 여기에 연결하려는 객체 사이에서 인터페이스 역할을 수행하는 패턴이다. • 네트워크 연결, 메모리의 대용량 객체로의 접근 등에 주로 이용한다.

행위 패턴(Behavioral Pattern)

행위 패턴은 클래스나 객체들이 서로 상호작용하는 방법이나 책임 분배 방법을 정의하는 패턴으로 총 11개의 패턴이 있다.

- 행위 패턴은 하나의 객체로 수행할 수 없는 작업을 여러 객체로 분배하면서 결합도를 최소화 할 수 있도록 도와준다.

책임 연쇄 (Chain of Responsibility)	<ul style="list-style-type: none"> • 요청을 처리할 수 있는 객체가 둘 이상 존재하여 한 객체가 처리하지 못하면 다음 객체로 넘어가는 형태의 패턴이다. • 요청을 처리할 수 있는 각 객체들이 고리(Chain)로 묶여 있어 요청이 해결될 때까지 고리를 따라 책임이 넘어간다.
커맨드 (Command)	<ul style="list-style-type: none"> • 요청을 객체의 형태로 캡슐화하여 재이용하거나 취소할 수 있도록 요청에 필요한 정보를 저장하거나 로그에 남기는 패턴이다. • 요청에 사용되는 각종 명령어들을 추상 클래스와 구체 클래스로 분리하여 단순화한다.
인터프리터 (Interpreter)	<ul style="list-style-type: none"> • 언어에 문법 표현을 정의하는 패턴이다. • SQL이나 통신 프로토콜과 같은 것을 개발할 때 사용한다.

반복자(Iterator)	<ul style="list-style-type: none"> 자료 구조와 같이 접근이 잦은 객체에 대해 동일한 인터페이스를 사용하도록 하는 패턴이다. 내부 표현 방법의 노출 없이 순차적인 접근이 가능하다.
중재자(Mediator)	<ul style="list-style-type: none"> 수많은 객체들 간의 복잡한 상호작용(Interface)을 캡슐화하여 객체로 정의하는 패턴이다. 객체 사이의 의존성을 줄여 결합도를 감소시킬 수 있다.
메멘토(Memento)	<ul style="list-style-type: none"> 특정 시점에서의 객체 내부 상태를 객체화함으로써 이후 요청에 따라 객체를 해당 시점의 상태로 돌릴 수 있는 기능을 제공하는 패턴이다. Ctrl+Z와 같은 되돌리기 기능을 개발할 때 주로 이용한다.
옵서버(Observer)	<ul style="list-style-type: none"> 한 객체의 상태가 변화하면 객체에 상속되어 있는 다른 객체들에게 변화된 상태를 전달하는 패턴이다. 주로 분산된 시스템 간에 이벤트를 생성·발행(Publish)하고, 이를 수신(Subscribe)해야 할 때 이용한다.
상태(State)	<ul style="list-style-type: none"> 객체의 상태에 따라 동일한 동작을 다르게 처리해야 할 때 사용하는 패턴이다. 객체 상태를 캡슐화하고 이를 참조하는 방식으로 처리한다.
전략(Strategy)	<ul style="list-style-type: none"> 동일한 계열의 알고리즘들을 개별적으로 캡슐화하여 상호 교환할 수 있게 정의하는 패턴이다. 클라이언트는 독립적으로 원하는 알고리즘을 선택하여 사용할 수 있으며, 클라이언트에 영향 없이 알고리즘의 변경이 가능하다.
템플릿 메소드(Template Method)	<ul style="list-style-type: none"> 상위 클래스에서 골격을 정의하고, 하위 클래스에서 세부 처리를 구체화하는 구조의 패턴이다. 유사한 서브 클래스를 묶어 공통된 내용을 상위 클래스에서 정의함으로써 코드의 양을 줄이고 유지보수를 용이하게 해준다.
방문자(Visitor)	<ul style="list-style-type: none"> 각 클래스들의 데이터 구조에서 처리 기능을 분리하여 별도의 클래스로 구성하는 패턴이다. 분리된 처리 기능은 각 클래스를 방문(Visit)하여 수행한다.

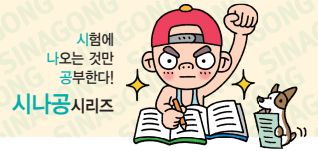
핵심 063 시스템 인터페이스 요구사항 구성

시스템 인터페이스는 독립적으로 떨어져 있는 시스템들 끼리 서로 연동하여 상호 작용하기 위한 접속 방법이나 규칙을 의미한다.

- 시스템 인터페이스 요구사항은 개발을 목표로 하는 시스템과 외부 시스템을 연동하는데 필요한 시스템 인터페이스에 대한 요구사항을 기술한 것이다.
- 시스템 인터페이스 요구사항 명세서의 구성 요소
 - 인터페이스 이름
 - 연계 대상 시스템
 - 연계 범위 및 내용
 - 연계 방식
 - 송신 데이터
 - 인터페이스 주기
 - 기타 고려사항 등

핵심 064 시스템 인터페이스 요구사항 분석 절차

- 소프트웨어 요구사항 목록에서 시스템 인터페이스 관련 요구사항을 선별하여 별도로 시스템 인터페이스 요구사항 목록을 만든다.
- 시스템 인터페이스와 관련된 요구사항 및 아키텍처 정의서, 현행 시스템의 대·내외 연계 시스템 현황 자료 등 시스템 인터페이스 요구사항과 관련된 자료를 준비한다.
- 시스템 인터페이스에 대한 요구사항 명세서를 확인하여 기능적인 요구사항과 비기능적인 요구사항으로 분류한다.
- 시스템 인터페이스 요구사항 명세서와 시스템 인터페이스 요구사항 목록 및 기타 관련 자료들을 비교하여 요구사항을 분석하고 내용을 추가하거나 수정한다.
- 추가·수정한 시스템 인터페이스 요구사항 명세서와 시스템 인터페이스 요구사항 목록을 관련 이해관계자에게 전달한다.



핵심 065 요구사항 검증 방법

- 요구사항 검토(Requirements Review) : 요구사항 명세서의 오류 확인 및 표준 준수 여부 등의 결함 여부를 검토 담당자들이 수작업으로 분석하는 방법으로, 동료검토, 워크스루, 인스펙션 등이 있다.

동료검토 (Peer Review)	요구사항 명세서 작성자가 명세서 내용을 직접 설명하고 동료들이 이를 들으면서 결함을 발견하는 형태의 검토 방법이다.
워크스루 (Walk Through)	검토 회의 전에 요구사항 명세서를 미리 배포하여 사전 검토한 후에 짧은 검토 회의를 통해 결함을 발견하는 형태의 검토 방법이다.
인스펙션 (Inspection)	요구사항 명세서 작성자를 제외한 다른 검토 전문가들이 요구사항 명세서를 확인하면서 결함을 발견하는 형태의 검토 방법이다.

- 프로토타이핑(Prototyping) : 사용자의 요구사항을 정확히 파악하기 위해 실제 개발될 소프트웨어에 대한 견본품(Prototype)을 만들어 최종 결과물을 예측한다.
- 테스트 설계 : 요구사항은 테스트할 수 있도록 작성되어야 하며, 이를 위해 테스트 케이스(Test Case)를 생성하여 이후에 요구사항이 현실적으로 테스트 가능한지를 검토한다.
- CASE 도구 활용 : 일관성 분석(Consistency Analysis)을 통해 요구사항 변경사항의 추적 및 분석, 관리하고, 표준 준수 여부를 확인한다.

핵심 066 인터페이스 요구사항 검증의 주요 항목

완전성 (Completeness)	사용자의 모든 요구사항이 누락되지 않고 완전하게 반영되어 있는가?
일관성 (Consistency)	요구사항이 모순되거나 충돌되는 점 없이 일관성을 유지하고 있는가?
명확성 (Unambiguity)	모든 참여자가 요구사항을 명확히 이해할 수 있는가?
기능성 (Functionality)	요구사항이 '어떻게(How to)' 보다 '무엇을(What)'에 중점을 두고 있는가?

검증 가능성 (Verifiability)	요구사항이 사용자의 요구를 모두 만족하고, 개발된 소프트웨어가 사용자의 요구 내용과 일치하는지를 검증할 수 있는가?
추적 가능성 (Traceability)	요구사항 명세서와 설계서를 추적할 수 있는가?
변경 용이성 (Easily Changeable)	요구사항 명세서의 변경이 쉽도록 작성되었는가?

핵심 067 인터페이스 시스템 식별

개발 시스템 식별	인터페이스 관련 자료들을 기반으로 개발하고자 하는 시스템의 상세 식별 정보를 정의하고 목록을 작성하는 것이다.
내·외부 시스템 식별	인터페이스 관련 자료들을 기반으로 개발할 시스템과 연계할 내·외부 시스템들의 상세 식별 정보를 정의하고 목록을 작성하는 것이다.
내·외부 시스템 환경 및 관리 주체 식별	<ul style="list-style-type: none"> • 내·외부 시스템 환경 : 연계할 시스템 접속에 필요한 IP 또는 URL, Port 정보 등 시스템의 실제 운용 환경을 의미한다. • 내·외부 시스템 관리 주체 : 하드웨어를 실제적으로 관리하는 담당자를 의미한다. • 인터페이스 관련 자료들을 기반으로 내·외부 시스템의 실제 운용 환경과 하드웨어 관리 주체를 확인한다.
내·외부 시스템 네트워크 연결 정보 식별	<ul style="list-style-type: none"> • 인터페이스 관련 자료들을 기반으로 내·외부 시스템을 연계하는데 필요한 네트워크 연결 정보를 확인한다. • 내·외부 시스템 네트워크 연결 정보는 시스템 로그인 및 DB 정보를 의미한다.
인터페이스 식별	인터페이스 요구사항 명세서와 인터페이스 요구사항 목록을 기반으로 개발할 시스템과 이와 연계할 내·외부 시스템 사이의 인터페이스를 식별하고 인터페이스 목록을 작성하는 것이다.
인터페이스 시스템 식별	인터페이스별로 인터페이스에 참여하는 시스템들을 송신 시스템과 수신 시스템으로 구분하여 작성하는 것이다.



핵심 068 식별 대상 데이터

- 식별 대상 데이터는 송·수신 시스템 사이에서 교환되는 데이터로, 규격화된 표준 형식에 따라 전송된다.
- 인터페이스 표준 항목
 - 인터페이스 표준 항목은 송·수신 시스템을 연계하는데 표준적으로 필요한 데이터를 의미한다.
 - 인터페이스 표준 항목은 시스템 공통부와 거래 공통부로 나뉜다.

시스템 공통부	<ul style="list-style-type: none"> • 시스템 간 연동 시 필요한 공통 정보이다. • 구성 정보 : 인터페이스 ID, 전송 시스템 정보, 서비스 코드 정보, 응답 결과 정보, 장애 정보 등
거래 공통부	<ul style="list-style-type: none"> • 시스템들이 연동된 후 송·수신 되는 데이터를 처리할 때 필요한 정보이다. • 구성 정보 : 직원 정보, 승인자 정보, 기기 정보, 매체 정보 등

- 송·수신 데이터 항목
 - 송·수신 데이터 항목은 송·수신 시스템이 업무를 수행하는 데 사용하는 데이터이다.
 - 전송되는 데이터 항목과 순서는 인터페이스별로 다르다.
- 공통 코드
 - 공통 코드는 시스템들에서 공통적으로 사용하는 코드이다.
 - 연계 시스템이나 연계 소프트웨어에서 사용하는 상태 및 오류 코드 등과 같은 항목에 대해 코드값과 코드명, 코드 설명 등을 공통 코드로 관리한다.

핵심 069 송·수신 데이터 식별

개발할 시스템과 연계할 내·외부 시스템 사이의 정보 흐름과 데이터베이스 산출물을 기반으로 송·수신 데이터를 식별한다.

- 인터페이스 표준 항목과 송·수신 데이터 항목 식별 : 송·수신 시스템 사이의 교환 범위를 확인하고 인터페이스 표준 항목에 대해 송·수신 데이터 항목을 식별한다.
- 코드성 데이터 항목 식별 : 코드성 데이터 항목에 대해 코드, 코드명, 코드 설명 등의 코드 정보를 식별한다.

- 코드성 데이터 항목에 대해 송신 시스템에서 사용하는 코드 정보와 수신 시스템에서 사용하는 코드 정보가 동일한 경우 공통 코드 정보를 확보하고, 다른 경우 매핑 필요 대상으로 분류하여 양쪽 시스템에서 사용하는 코드 정보를 확보한다.

핵심 070 시스템 연계 기술

DB Link	DB에서 제공하는 DB Link 객체를 이용하는 방식이다.
API/Open API	송신 시스템의 데이터베이스(DB)에서 데이터를 읽어와 제공하는 애플리케이션 프로그래밍 인터페이스 프로그램이다.
연계 솔루션	EAI 서버와 송·수신 시스템에 설치되는 클라이언트(Client)를 이용하는 방식이다.
Socket	서버는 통신을 위한 소켓(Socket)을 생성하여 포트를 할당하고 클라이언트의 통신 요청 시 클라이언트와 연결하여 통신하는 네트워크 기술이다.
Web Service	웹 서비스(Web Service)에서 WSDL과 UDDI, SOAP 프로토콜을 이용하여 연계하는 서비스이다.

핵심 071 인터페이스 통신 유형 / 처리 유형 / 발생 주기

인터페이스 통신 유형

단방향	시스템에서 거래를 요청만 하고 응답이 없는 방식이다.
동기	시스템에서 거래를 요청하고 응답이 올 때까지 대기(Request-Reply)하는 방식이다.
비동기	시스템에서 거래를 요청하고 다른 작업을 수행하다 응답이 오면 처리하는 방식(Send-Receive, Send-Receive-Acknowledge, Publish-Subscribe)이다.

인터페이스 처리 유형

실시간 방식	사용자가 요청한 내용을 바로 처리해야 할 때 사용하는 방식이다.
지연 처리 방식	데이터를 매건 단위로 처리할 경우 비용이 많이 발생할 때 사용하는 방식이다.
배치 방식	대량의 데이터를 처리할 때 사용하는 방식이다.

인터페이스 발생 주기

인터페이스 발생 주기는 업무의 성격과 송·수신 데이터 전송량을 고려하여 매일, 수시, 주 1회 등으로 구분한다.

핵심 072 미들웨어(Middleware)

미들웨어는 미들(Middle)과 소프트웨어(Software)의 합성어로, 운영체제와 응용 프로그램, 또는 서버와 클라이언트 사이에서 다양한 서비스를 제공하는 소프트웨어이다.

DB(DataBase)	<ul style="list-style-type: none"> DB는 데이터베이스 벤더에서 제공하는 클라이언트에서 원격의 데이터베이스와 연결하기 위한 미들웨어이다. DB를 사용하여 시스템을 구축하는 경우 보통 2-Tier 아키텍처라고 한다.
RPC(Remote Procedure Call)	RPC(원격 프로시저 호출)는 응용 프로그램의 프로시저를 사용하여 원격 프로시저를 마치 로컬 프로시저처럼 호출하는 방식의 미들웨어이다.
MOM(Message Oriented Middleware)	<ul style="list-style-type: none"> MOM(메시지 지향 미들웨어)은 메시지 기반의 비동기형 메시지를 전달하는 방식의 미들웨어이다. 온라인 업무보다는 이기종 분산 데이터 시스템의 데이터 동기를 위해 많이 사용된다.
TP-Monitor (Transaction Processing Monitor)	<ul style="list-style-type: none"> TP-Monitor(트랜잭션 처리 모니터)는 항공기나 철도 예약 업무 등과 같은 온라인 트랜잭션 업무에서 트랜잭션을 처리 및 감시하는 미들웨어이다. 사용자 수가 증가해도 빠른 응답 속도를 유지해야 하는 업무에 주로 사용된다.
ORB(Object Request Broker)	<ul style="list-style-type: none"> ORB(객체 요청 브로커)는 객체 지향 미들웨어로 코바(CORBA) 표준 스펙을 구현한 미들웨어이다. 최근에는 TP-Monitor의 장점인 트랜잭션 처리와 모니터링 등을 추가로 구현한 제품도 있다.
WAS(Web Application Server)	<ul style="list-style-type: none"> WAS(웹 애플리케이션 서버)는 정적인 콘텐츠를 처리하는 웹 서버와 달리 사용자의 요구에 따라 변하는 동적인 콘텐츠를 처리하기 위해 사용되는 미들웨어이다. 클라이언트/서버 환경보다는 웹 환경을 구현하기 위한 미들웨어이다.

불합격 방지용 안전장치 기억상자

틀린 문제만 모아 오답 노트를 만들고 싶다고요?
꺼먹기 전에 다시 한 번 복습하고 싶다고요?
지금 당장 QR 코드를 스캔해 보세요.



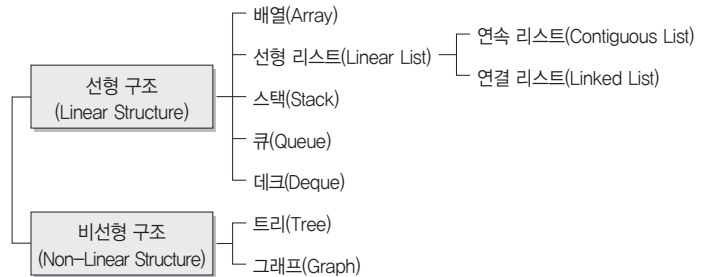
2 과목 소프트웨어 개발

핵심 073 자료 구조

자료 구조의 정의

효율적인 프로그램을 작성할 때 가장 우선적인 고려사항은 저장 공간의 효율성과 실행시간의 신속성이다. 자료 구조는 프로그램에서 사용하기 위한 자료를 기억장치의 공간 내에 저장하는 방법과 저장된 그룹 내에 존재하는 자료 간의 관계, 처리 방법 등을 연구 분석하는 것을 말한다.

자료 구조의 분류



핵심 074 배열(Array)

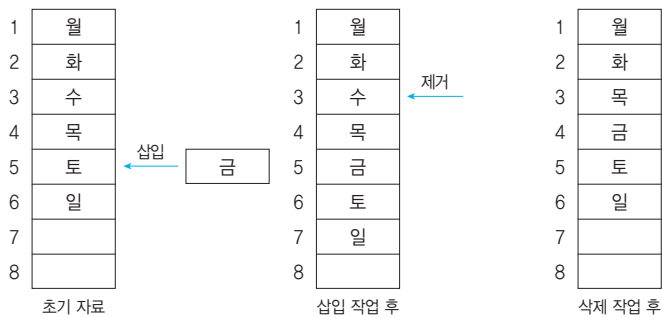
배열은 동일한 자료형의 데이터들이 같은 크기로 나열되어 순서를 갖고 있는 집합이다.

- 배열은 정적인 자료 구조로 기억장소의 추가가 어렵고, 데이터 삭제 시 데이터가 저장되어 있던 기억장소는 빈 공간으로 남아있어 메모리의 낭비가 발생한다.
- 배열은 첨자를 이용하여 데이터에 접근한다.
- 배열은 반복적인 데이터 처리 작업에 적합한 구조이다.
- 배열은 데이터마다 동일한 이름의 변수를 사용하여 처리가 간편하다.
- 배열은 사용한 첨자의 개수에 따라 n차원 배열이라고 부른다.

핵심 075 선형 리스트(Linear List)

• 연속 리스트(Contiguous List)

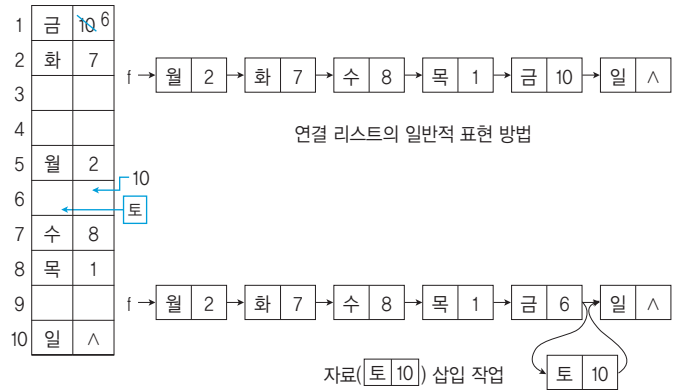
- 연속 리스트는 배열과 같이 연속되는 기억장소에 저장되는 자료 구조이다.
- 연속 리스트는 기억장소를 연속적으로 배정받기 때문에 기억장소 이용 효율은 밀도가 1로서 가장 좋다.
- 연속 리스트는 중간에 데이터를 삽입하기 위해서는 연속된 빈 공간이 있어야 하며, 삽입·삭제 시 자료의 이동이 필요하다.



• 연결 리스트(Linked List)

- 연결 리스트는 자료들을 반드시 연속적으로 배열시키지는 않고 임의의 기억공간에 기억시키되, 자료 항목의 순서에 따라 노드의 포인터 부분을 이용하여 서로 연결시킨 자료 구조이다.
- 연결 리스트는 노드의 삽입·삭제 작업이 용이하다.
- 기억 공간이 연속적으로 놓여 있지 않아도 저장할 수 있다.
- 연결 리스트는 연결을 위한 링크(포인터) 부분이 필요하기 때문에 순차 리스트에 비해 기억 공간의 이용 효율이 좋지 않다.
- 연결 리스트는 연결을 위한 포인터를 찾는 시간이 필요하기 때문에 접근 속도가 느리다.
- 연결 리스트는 중간 노드 연결이 끊어지면 그 다음 노드를 찾기 힘들다.

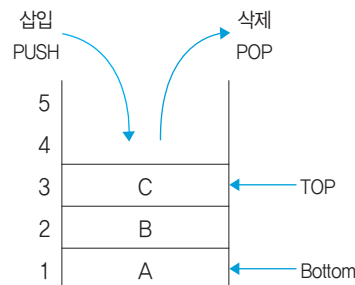
예 연결 리스트 기억장치 내에서의 표현 방법



핵심 076 스택(Stack)

스택은 리스트의 한쪽 끝으로만 자료의 삽입, 삭제 작업이 이루어지는 자료 구조이다.

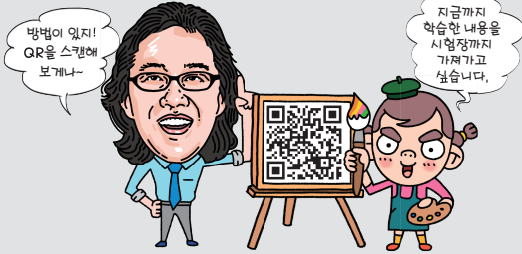
- 스택은 가장 나중에 삽입된 자료가 가장 먼저 삭제되는 후입선출(LIFO; Last In First Out) 방식으로 자료를 처리한다.
- 스택의 모든 기억 공간이 꽉 채워져 있는 상태에서 데이터가 삽입되면 오버플로(Overflow)가 발생하며, 더 이상 삭제할 데이터가 없는 상태에서 데이터를 삭제하면 언더플로(Underflow)가 발생한다.



- TOP : 스택으로 할당된 기억 공간에 가장 마지막으로 삽입된 자료가 기억된 위치를 가리키는 요소이다.
- Bottom : 스택의 가장 밑바닥이다.

불합격 방지용 안전장치 기억상자

틀린 문제만 모아 오답 노트를 만들고 싶다고요? 까먹기 전에 다시 한 번 복습하고 싶다고요? 지금까지 공부한 내용을 안전하게 시험장까지 가져가는 완벽한 방법이 있습니다. 지금 당장 QR 코드를 스캔해 보세요.

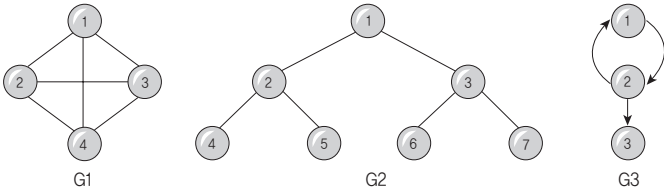


www.membox.co.kr을 직접 입력해도 접속할 수 있습니다.

핵심 077 그래프(Graph)

그래프 G는 정점 V(Vertex)와 간선 E(Edge)의 두 집합으로 이루어진다.

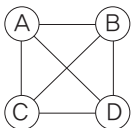
- 간선의 방향성 유무에 따라 방향 그래프와 무방향 그래프로 구분된다.
- 통신망(Network), 교통망, 이항관계, 연립방정식, 유기화학 구조식, 무향선분 해법 등에 응용된다.



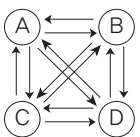
※ 방향/무방향 그래프의 최대 간선 수

n개의 정점으로 구성된 무방향 그래프에서 최대 간선 수는 $n(n-1)/2$ 이고, 방향 그래프에서 최대 간선 수는 $n(n-1)$ 이다.

예 정점이 4개인 경우 무방향 그래프와 방향 그래프의 최대 간선 수는 다음과 같다.



무방향 그래프의 최대 간선 수 : $4(4-1)/2 = 6$

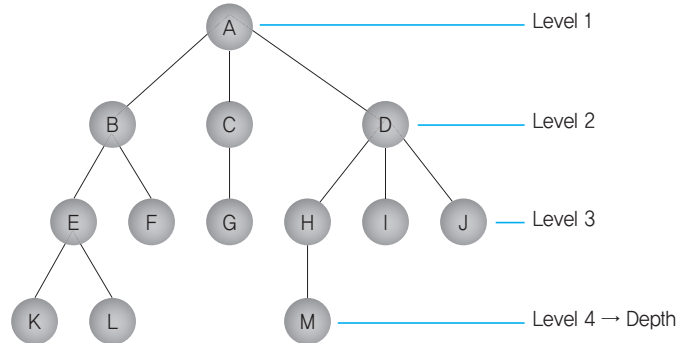


방향 그래프의 최대 간선 수 : $4(4-1) = 12$

핵심 078 트리의 개요

트리는 정점(Node, 노드)과 선분(Branch, 가지)을 이용하여 사이클을 이루지 않도록 구성된 그래프(Graph)의 특수한 형태이다.

- 트리는 가족의 계보(족보), 조직도 등을 표현하기에 적합하다.
- 트리 관련 용어



- 노드(Node) : 트리의 기본 요소로서 자료 항목과 다른 항목에 대한 가지(Branch)를 합친 것

예 A, B, C, D, E, F, G, H, I, J, K, L, M

- 근 노드(Root Node) : 트리의 맨 위에 있는 노드

예 A

- 디그리(Degree, 차수) : 각 노드에서 뻗어 나온 가지의 수

예 A = 3, B = 2, C = 1, D = 3

- 단말 노드(Terminal Node) = 잎 노드(Leaf Node) : 자식이 하나도 없는 노드, 즉 디그리가 0인 노드

예 K, L, F, G, M, I, J

- 자식 노드(Son Node) : 어떤 노드에 연결된 다음 레벨의 노드들

예 D의 자식 노드 : H, I, J

- 부모 노드(Parent Node) : 어떤 노드에 연결된 이전 레벨의 노드들

예 E, F의 부모 노드 : B

- 형제 노드(Brother Node, Sibling) : 동일한 부모를 갖는 노드들

예 H의 형제 노드 : I, J

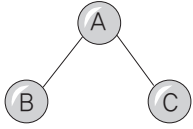
- 트리의 디그리 : 노드들의 디그리 중에서 가장 많은 수

예 노드 A나 D가 3개의 디그리를 가지므로 앞 트리의 디그리는 3이다.



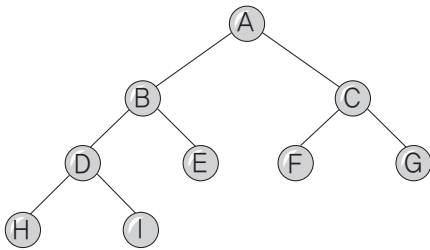
핵심 079 트리의 운행법

이진 트리의 운행법은 다음 세 가지가 있다.

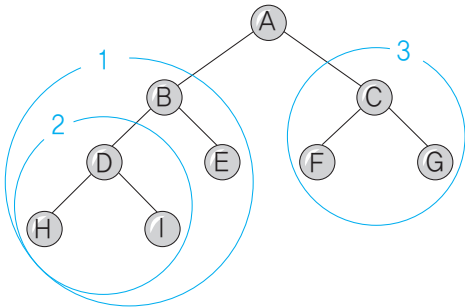


- Preorder 운행 : Root → Left → Right 순으로 운행한다. A, B, C
- Inorder 운행 : Left → Root → Right 순으로 운행한다. B, A, C
- Postorder 운행 : Left → Right → Root 순으로 운행한다. B, C, A

예제 다음 트리를 Inorder, Preorder, Postorder 방법으로 운행했을 때 각 노드를 방문한 순서는?



Preorder 운행법의 방문 순서



- 1 Preorder는 Root → Left → Right이므로 A13이 된다.
 - 2 1은 B2E이므로 AB2E3이 된다.
 - 3 2는 DHI이므로 ABDHIE3이 된다.
 - 4 3은 CFG이므로 ABDHIECFG가 된다.
- 방문 순서 : ABDHIECFG

Inorder 운행법의 방문 순서

- 1 Inorder는 Left → Root → Right이므로 1A3이 된다.
- 2 1은 2BE이므로 2BEA3이 된다.

- 3 2는 HDI이므로 HDIBEA3이 된다.
- 4 3은 FCG이므로 HDIBEAFCG가 된다.

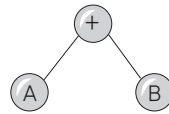
• 방문 순서 : HDIBEAFCG

Postorder 운행법의 방문 순서

- 1 Postorder는 Left → Right → Root이므로 13A가 된다.
 - 2 1은 2EB이므로 2EB3A가 된다.
 - 3 2는 HID이므로 HIDEB3A가 된다.
 - 4 3은 FGC이므로 HIDEBFGCA가 된다.
- 방문 순서 : HIDEBFGCA

핵심 080 수식의 표기법

산술식을 계산하기 위해 기억공간에 기억시키는 방법으로 이진 트리를 많이 사용한다. 이진 트리로 만들어진 수식을 인오더, 프리오더, 포스트오더로 운행하면 각각 중위(Infix), 전위(Prefix), 후위(Postfix) 표기법이 된다.



- 전위 표기법(Prefix) : 연산자 → Left → Right, +AB
- 중위 표기법(Infix) : Left → 연산자 → Right, A+B
- 후위 표기법(Postfix) : Left → Right → 연산자, AB+

Infix 표기를 Postfix나 Prefix로 바꾸기

- Postfix나 Prefix는 스택을 이용하여 처리하므로 Infix는 Postfix나 Prefix로 바꾸어 처리한다.

예제 1 다음과 같이 Infix로 표기된 수식을 Prefix와 Postfix로 변환하시오.

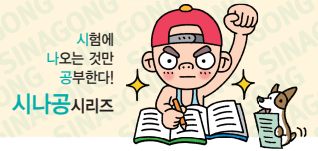
$$X = A / B * (C + D) + E$$

Prefix로 변환하기

- 1 연산 우선순위에 따라 괄호로 묶는다.
- 2 연산자를 해당 괄호의 앞(왼쪽)으로 옮긴다.

$$X = (((A / B) * (C + D)) + E)$$

$$\rightarrow = (X + (* (/ (AB) + (CD)) E))$$



③ 필요없는 괄호를 제거한다.

prefix 표기 : $= X + * / A B + C D E$

• Postfix로 변환하기

① 연산 우선순위에 따라 괄호로 묶는다.

$$(X = ((A / B) * (C + D)) + E)$$

② 연산자를 해당 괄호의 뒤(오른쪽)로 옮긴다.

$$(X = ((A / B) * (C + D)) + E) \rightarrow (X ((A B) / (C D) +) * E) + =$$

③ 필요없는 괄호를 제거한다.

Postfix 표기 : $X A B / C D + * E + =$

Postfix나 Prefix로 표기된 수식을 Infix로 바꾸기

예제 2 다음과 같이 Postfix로 표기된 수식을 Infix로 변환하시오.

$A B C - / D E F + * +$

• Postfix는 Infix 표기법에서 연산자를 해당 피연산자 두 개의 뒤로 이동한 것이므로 연산자를 다시 해당 피연산자 두 개의 가운데로 옮기면 된다.

① 먼저 인접한 피연산자 두 개와 오른쪽의 연산자를 괄호로 묶는다.

$$((A (B C -) /) (D (E F +) *) +)$$

② 연산자를 해당 피연산자의 가운데로 이동시킨다.

$$((A (B C -) /) (D (E F +) *) +) \rightarrow ((A / (B - C)) + (D * (E + F)))$$

③ 필요 없는 괄호를 제거한다.

$$((A / (B - C)) + (D * (E + F))) \rightarrow A / (B - C) + D * (E + F)$$

예제 3 다음과 같이 Prefix로 표기된 수식을 Infix로 변환하시오.

$+ / A - B C * D + E F$

• Prefix는 Infix 표기법에서 연산자를 해당 피연산자 두 개의 앞으로 이동한 것이므로 연산자를 다시 해당 피연산자 두 개의 가운데로 옮기면 된다.

① 먼저 인접한 피연산자 두 개와 왼쪽의 연산자를 괄호로 묶는다.

$$(+ (/ A (- B C)) (* D (+ E F)))$$

② 연산자를 해당 피연산자 사이로 이동시킨다.

$$(+ (/ A (- B C)) (* D (+ E F))) \rightarrow ((A / (B - C)) + (D * (E + F)))$$

③ 필요 없는 괄호를 제거한다.

$$((A / (B - C)) + (D * (E + F))) \rightarrow A / (B - C) + D * (E + F)$$

핵심 081 삽입 정렬(Insertion Sort)

삽입 정렬은 가장 간단한 정렬 방식으로 이미 순서화된 파일에 새로운 하나의 레코드를 순서에 맞게 삽입시켜 정렬한다.

• 두 번째 키와 첫 번째 키를 비교해 순서대로 나열(1회전)하고, 이어서 세 번째 키를 첫 번째, 두 번째 키와 비교해 순서대로 나열(2회전)하고, 계속해서 n번째 키를 앞의 n-1개의 키와 비교하여 알맞은 순서에 삽입하여 정렬하는 방식이다.

• 평균과 최악 모두 수행 시간 복잡도는 $O(n^2)$ 이다.

예제 8, 5, 6, 2, 4를 삽입 정렬로 정렬하시오.

• 초기 상태:

8	5	6	2	4
---	---	---	---	---

• 1회전:

8	5	6	2	4
---	---	---	---	---

 →

5	8	6	2	4
---	---	---	---	---

두 번째 값을 첫 번째 값과 비교하여 5를 첫 번째 자리에 삽입하고 8을 한 칸 뒤로 이동시킨다.

• 2회전:

5	8	6	2	4
---	---	---	---	---

 →

5	6	8	2	4
---	---	---	---	---

세 번째 값을 첫 번째, 두 번째 값과 비교하여 6을 8자리에 삽입하고 8은 한 칸 뒤로 이동시킨다.

• 3회전:

5	6	8	2	4
---	---	---	---	---

 →

2	5	6	8	4
---	---	---	---	---

네 번째 값 2를 처음부터 비교하여 맨 처음에 삽입하고 나머지를 한 칸씩 뒤로 이동시킨다.

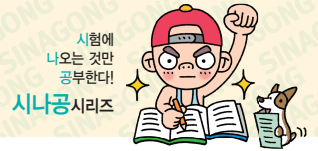
• 4회전:

2	5	6	8	4
---	---	---	---	---

 →

2	4	5	6	8
---	---	---	---	---

다섯 번째 값 4를 처음부터 비교하여 5자리에 삽입하고 나머지를 한 칸씩 뒤로 이동시킨다.



핵심 082 선택 정렬(Selection Sort)

선택 정렬은 n 개의 레코드 중에서 최소값을 찾아 첫 번째 레코드 위치에 놓고, 나머지 $(n-1)$ 개 중에서 다시 최소값을 찾아 두 번째 레코드 위치에 놓는 방식을 반복하여 정렬하는 방식이다.

- 평균과 최악 모두 수행 시간 복잡도는 $O(n^2)$ 이다.

예제 8, 5, 6, 2, 4를 선택 정렬로 정렬하시오.

• 초기 상태: [8 5 6 2 4]

• 1회전: [5 8 6 2 4] → [5 8 6 2 4] → [2 8 6 5 4] → [2 8 6 5 4]

• 2회전: [2 6 8 5 4] → [2 5 8 6 4] → [2 4 8 6 5]

• 3회전: [2 4 6 8 5] → [2 4 5 8 6]

• 4회전: [2 4 5 6 8]

핵심 083 버블 정렬(Bubble Sort)

버블 정렬은 주어진 파일에서 인접한 두 개의 레코드 키 값을 비교하여 그 크기에 따라 레코드 위치를 서로 교환하는 정렬 방식이다.

- 계속 정렬 여부를 플래그 비트(f)로 결정한다.
- 평균과 최악 모두 수행 시간 복잡도는 $O(n^2)$ 이다.

예제 8, 5, 6, 2, 4를 버블 정렬로 정렬하시오.

• 초기 상태: [8 5 6 2 4]

• 1회전: [5 8 6 2 4] → [5 6 8 2 4] → [5 6 2 8 4] → [5 6 2 4 8]

• 2회전: [5 6 2 4 8] → [5 2 6 4 8] → [5 2 4 6 8]

• 3회전: [2 5 4 6 8] → [2 4 5 6 8]

• 4회전: [2 4 5 6 8]

핵심 084 데이터베이스

데이터베이스는 특정 조직의 업무를 수행하는 데 필요한 상호 관련된 데이터들의 모임으로 다음과 같이 정의할 수 있다.

- 통합된 데이터(Integrated Data) : 자료의 중복을 배제한 데이터의 모임이다.
- 저장된 데이터(Stored Data) : 컴퓨터가 접근할 수 있는 저장 매체에 저장된 자료이다.
- 운영 데이터(Operational Data) : 조직의 고유한 업무를 수행하는 데 존재 가치가 확실하고 없어서는 안 될 반드시 필요한 자료이다.
- 공용 데이터(Shared Data) : 여러 응용 시스템들이 공동으로 소유하고 유지하는 자료이다.

핵심 085 DBMS(DataBase Management System; 데이터베이스 관리 시스템)

DBMS란 사용자와 데이터베이스 사이에서 사용자의 요구에 따라 정보를 생성해주고, 데이터베이스를 관리해주는 소프트웨어이다.

- DBMS는 기존의 파일 시스템이 갖는 데이터의 종속성과 중복성의 문제를 해결하기 위해 제한된 시스템으로, 모든 응용 프로그램들이 데이터베이스를 공유할 수 있도록 관리해 준다.
- DBMS의 필수 기능에는 정의(Definition), 조작(Manipulation), 제어(Control) 기능이 있다.
 - 정의 기능 : 모든 응용 프로그램들이 요구하는 데이터 구조를 지원하기 위해 데이터베이스에 저장될 데이터의 형(Type)과 구조에 대한 정의, 이용 방식, 제약 조건 등을 명시하는 기능이다.
 - 조작 기능 : 데이터 검색, 갱신, 삽입, 삭제 등을 체계적으로 처리하기 위해 사용자와 데이터베이스 사이의 인터페이스 수단을 제공하는 기능이다.

- 제어 기능

- ▶ 데이터베이스를 접근하는 갱신, 삽입, 삭제 작업이 정확하게 수행되어 데이터의 무결성이 유지되도록 제어해야 한다.
- ▶ 정당한 사용자가 허가된 데이터만 접근할 수 있도록 보안(Security)을 유지하고 권한(Authority)을 검사할 수 있어야 한다.
- ▶ 여러 사용자가 데이터베이스를 동시에 접근하여 데이터를 처리할 때 처리 결과가 항상 정확성을 유지하도록 병행 제어(Concurrency Control)를 할 수 있어야 한다.

핵심 086 DBMS의 장·단점

장점	<ul style="list-style-type: none"> • 데이터의 논리적, 물리적 독립성이 보장된다. • 데이터의 중복을 피할 수 있어 기억 공간이 절약된다. • 저장된 자료를 공동으로 이용할 수 있다. • 데이터의 일관성을 유지할 수 있다. • 데이터의 무결성을 유지할 수 있다. • 보안을 유지할 수 있다. • 데이터를 표준화할 수 있다. • 데이터를 통합하여 관리할 수 있다. • 항상 최신의 데이터를 유지한다. • 데이터의 실시간 처리가 가능하다.
단점	<ul style="list-style-type: none"> • 데이터베이스의 전문가가 부족하다. • 전산화 비용이 증가한다. • 대용량 디스크로의 집중적인 Access로 과부하(Overhead)가 발생한다. • 파일의 예비(Backup)와 회복(Recovery)이 어렵다. • 시스템이 복잡하다.

핵심 087 스키마

스키마(Schema)는 데이터베이스의 구조와 제약 조건에 관한 전반적인 명세(Specification)를 기술(Description)한 메타데이터(Meta-Data)의 집합이다.

- 스키마는 데이터베이스를 구성하는 데이터 개체(Entity), 속성(Attribute), 관계(Relationship) 및 데이터 조작 시 데이터 값들이 갖는 제약 조건 등에 관해 전반적으로 정의한다.

- 스키마는 사용자의 관점에 따라 외부 스키마, 개념 스키마, 내부 스키마로 나누어진다.

외부 스키마	사용자나 응용 프로그래머가 각 개인의 입장에서 필요로 하는 데이터베이스의 논리적 구조를 정의한 것이다.
개념 스키마	데이터베이스의 전체적인 논리적 구조로서, 모든 응용 프로그램이나 사용자들이 필요로 하는 데이터를 종합한 조직 전체의 데이터베이스로 하나만 존재한다.
내부 스키마	물리적 저장장치의 입장에서 본 데이터베이스 구조로서, 실제로 데이터베이스에 저장될 레코드의 형식을 정의하고 저장 데이터 항목의 표현 방법, 내부 레코드의 물리적 순서 등을 나타낸다.

핵심 088 데이터 접속(Data Mapping)

데이터 접속은 소프트웨어의 기능 구현을 위해 프로그래밍 코드와 데이터베이스의 데이터를 연결(Mapping)하는 것을 말하며, 관련 기술로 SQL Mapping과 ORM이 있다.

- SQL Mapping : 프로그래밍 코드 내에 SQL을 직접 입력하여 DBMS의 데이터에 접속하는 기술로, 관련 프레임워크에는 JDBC, ODBC, MyBatis 등이 있다.
- ORM(Object-Relational Mapping) : 객체지향 프로그래밍의 객체(Object)와 관계형(Relational) 데이터베이스의 데이터를 연결(Mapping)하는 기술로, 관련 프레임워크에는 JPA, Hibernate, Django 등이 있다.

핵심 089 트랜잭션(Transaction)

트랜잭션은 데이터베이스의 상태를 변환시키는 하나의 논리적 기능을 수행하기 위한 작업의 단위 또는 한꺼번에 모두 수행되어야 할 일련의 연산들을 의미한다.

- 트랜잭션을 제어하기 위해서 사용하는 명령어를 TCL(Transaction Control Language)이라고 하며, TCL의 종류에는 COMMIT, ROLLBACK, SAVEPOINT가 있다.

- COMMIT : 트랜잭션 처리가 정상적으로 종료되어 트랜잭션이 수행한 변경 내용을 데이터베이스에 반영하는 명령어
- ROLLBACK : 하나의 트랜잭션 처리가 비정상적으로 종료되어 데이터베이스의 일관성이 깨졌을 때 트랜잭션이 행한 모든 변경 작업을 취소하고 이전 상태로 되돌리는 연산
- SAVEPOINT(=CHECKPOINT) : 트랜잭션 내에 ROLLBACK 할 위치인 저장점을 지정하는 명령어

핵심 090 절차형 SQL의 개요

절차형 SQL은 C, JAVA 등의 프로그래밍 언어와 같이 연속적인 실행이나 분기, 반복 등의 제어가 가능한 SQL을 의미한다.

- 절차형 SQL은 일반적인 프로그래밍 언어에 비해 효율은 떨어지지만 단일 SQL 문장으로 처리하기 어려운 연속적인 작업들을 처리하는데 적합하다.
- 절차형 SQL을 활용하여 다양한 기능을 수행하는 저장 모듈을 생성할 수 있다.
- 절차형 SQL은 DBMS 엔진에서 직접 실행되기 때문에 입·출력 패킷이 적은 편이다.
- BEGIN ~ END 형식으로 작성되는 블록(Block) 구조로 되어 있기 때문에 기능별 모듈화가 가능하다.
- 절차형 SQL의 종류에는 프로시저, 트리거, 사용자 정의 함수가 있다.
 - 프로시저(Procedure) : 특정 기능을 수행하는 일종의 트랜잭션 언어로, 호출을 통해 실행되어 미리 저장해 놓은 SQL 작업을 수행한다.
 - 트리거(Trieger) : 데이터베이스 시스템에서 데이터의 입력, 갱신, 삭제 등의 이벤트(Event)가 발생할 때마다 관련 작업이 자동으로 수행된다.
- 사용자 정의 함수 : 프로시저와 유사하게 SQL을 사용하여 일련의 작업을 연속적으로 처리하며, 종료 시 예약어 Return을 사용하여 처리 결과를 단일값으로 반환한다.

핵심 091 절차형 SQL의 테스트와 디버깅

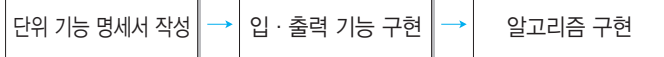
절차형 SQL은 디버깅을 통해 기능의 적합성 여부를 검증하고, 실행을 통해 결과를 확인하는 테스트 과정을 수행한다.

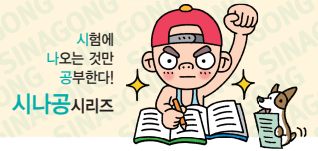
- 절차형 SQL은 테스트 전에 생성을 통해 구문 오류(Syntax Error)나 참조 오류의 존재 여부를 확인한다.
- 많은 코드로 구성된 절차형 SQL의 특성상 오류 및 경고 메시지가 상세히 출력되지 않으므로 SHOW 명령어를 통해 내용을 확인하고 문제를 수정한다.
- 정상적으로 생성된 절차형 SQL은 디버깅을 통해 로직을 검증하고, 결과를 통해 최종적으로 확인한다.
- 절차형 SQL의 디버깅은 실제로 데이터베이스에 변화를 줄 수 있는 삽입 및 변경 관련 SQL문을 주석으로 처리하고, 출력문을 이용하여 화면에 출력하여 확인한다.

핵심 092 단위 모듈(Unit Module)의 개요

단위 모듈은 소프트웨어 구현에 필요한 여러 동작 중 한 가지 동작을 수행하는 기능을 모듈로 구현한 것이다.

- 단위 모듈로 구현되는 하나의 기능을 단위 기능이라고 부른다.
- 단위 모듈은 사용자나 다른 모듈로부터 값을 전달받아 시작되는 작은 프로그램을 의미하기도 한다.
- 두 개의 단위 모듈이 합쳐질 경우 두 개의 기능을 구현할 수 있다.
- 단위 모듈의 구성 요소에는 처리문, 명령문, 데이터 구조 등이 있다.
- 단위 모듈은 독립적인 컴파일이 가능하며, 다른 모듈에 호출되거나 삽입되기도 한다.
- 단위 모듈을 구현하기 위해서는 단위 기능 명세서를 작성한 후 입·출력 기능과 알고리즘을 구현해야 한다.





핵심 093 IPC(Inter-Process Communication)

IPC는 모듈 간 통신 방식을 구현하기 위해 사용되는 대표적인 프로그래밍 인터페이스 집합으로, 복수의 프로세스를 수행하며 이뤄지는 프로세스 간 통신까지 구현이 가능하다.

- IPC의 대표 메소드 5가지

Shared Memory	다수의 프로세스가 공유 가능한 메모리를 구성하여 프로세스 간 통신을 수행
Socket	네트워크 소켓을 이용하여 네트워크를 경유하는 프로세스들 간 통신을 수행
Semaphores	공유 자원에 대한 접근 제어를 통해 프로세스 간 통신을 수행
Pipes&named Pipes	<ul style="list-style-type: none"> • 'Pipe'라고 불리는 선입선출 형태로 구성된 메모리를 여러 프로세스가 공유하여 통신을 수행 • 하나의 프로세스가 Pipe를 이용 중이라면 다른 프로세스는 접근할 수 없음
Message Queueing	메시지가 발생하면 이를 전달하는 형태로 프로세스 간 통신을 수행

핵심 094 단위 모듈 테스트의 개요

단위 모듈 테스트는 프로그램의 단위 기능을 구현하는 모듈이 정해진 기능을 정확히 수행하는지 검증하는 것이다.

- 단위 모듈 테스트는 단위 테스트(Unit Test)라고도 하며, 화이트박스 테스트와 블랙박스 테스트 기법을 사용한다.
- 단위 모듈 테스트를 수행하기 위해서는 모듈을 단독적으로 실행할 수 있는 환경과 테스트에 필요한 데이터가 모두 준비되어야 한다.
- 모듈의 통합 이후에는 오랜 시간 추적해야 발견할 수 있는 에러들도 단위 모듈 테스트를 수행하면 쉽게 발견하고 수정할 수 있다.
- 단위 모듈 테스트의 기준은 단위 모듈에 대한 코드이므로 시스템 수준의 오류는 잡아낼 수 없다.

핵심 095 테스트 케이스(Test Case)

테스트 케이스는 구현된 소프트웨어가 사용자의 요구사항을 정확하게 준수했는지를 확인하기 위해 설계된 입력값, 실행 조건, 기대 결과 등으로 구성된 테스트 항목에 대한 명세서로, 명세 기반 테스트의 설계 산출물에 해당된다.

- 단위 모듈을 테스트하기 전에 테스트에 필요한 입력 데이터, 테스트 조건, 예상 결과 등을 모아 테스트 케이스를 만든다.
- 테스트 케이스를 이용하지 않고 수행하는 직관적인 테스트는 특정 요소에 대한 검증이 누락되거나 불필요한 검증의 반복으로 인해 인력과 시간을 낭비할 수 있다.
- ISO/IEC/IEEE 29119-3 표준에 따른 테스트 케이스의 구성 요소는 다음과 같다.

식별자(Identifier)	항목 식별자, 일련번호
테스트 항목(Test Item)	테스트 대상(모듈 또는 기능)
입력 명세(Input Specification)	입력 데이터 또는 테스트 조건
출력 명세(Output Specification)	테스트 케이스 수행 시 예상되는 출력 결과
환경 설정(Environmental Needs)	필요한 하드웨어나 소프트웨어의 환경
특수 절차 요구(Special Procedure Requirement)	테스트 케이스 수행 시 특별히 요구되는 절차
의존성 기술(Inter-case Dependencies)	테스트 케이스 간의 의존성

핵심 096 통합 개발 환경(IDE; Integrated Development Environment)

통합 개발 환경은 개발에 필요한 환경, 즉 편집기(Editor), 컴파일러(Compiler), 디버거(Debugger) 등의 다양한 툴을 하나의 인터페이스로 통합하여 제공하는 것을 의미한다.

- 통합 개발 환경 도구는 통합 개발 환경을 제공하는 소프트웨어를 의미한다.
- 통합 개발 환경 도구는 코드의 자동 생성 및 컴파일이 가능하고 추가 기능을 위한 도구들을 다운로드하여 추가할 수 있다.
- 통합 개발 환경을 지원하는 도구는 플랫폼, 운영체제, 언어별로 다양하게 존재하며, 대표적인 도구는 다음과 같다.

프로그램	개발사	플랫폼	운영체제	지원 언어
이클립스 (Eclipse)	Eclipse Foundation, IBM	크로스 플랫폼	Windows, Linux, MacOS 등	Java, C, C++, PHP, JSP 등
비주얼 스튜디오 (Visual Studio)	Microsoft	Win32, Win64	Windows	Basic, C, C++, C#, .NET 등
엑스 코드 (Xcode)	Apple	Mac, iPhone	MacOS, iOS	C, C++, C#, Java, AppleScript 등
안드로이드 스튜디오 (Android Studio)	Google	Android	Windows, Linux, MacOS	Java, C, C++
IDEA	JetBrains (이전 IntelliJ)	크로스 플랫폼	Windows, Linux, MacOS	Java, JSP, XML, Go, Kotlin, PHP 등

핵심 097 기타 협업 도구

협업 도구는 개발에 참여하는 사람들이 서로 다른 작업 환경에서 원활히 프로젝트를 수행할 수 있도록 도와주는 도구(Tool)로, 협업 소프트웨어, 그룹웨어(Groupware) 등으로도 불린다.

- 협업 도구에는 일정 관리, 업무흐름 관리, 정보 공유, 커뮤니케이션 등의 업무 보조 도구가 포함되어 있다.
- 협업 도구는 웹 기반, PC, 스마트폰 등 다양한 플랫폼에서 사용할 수 있도록 제공된다.
- 협업 도구에 익숙하지 않거나 이용할 의지가 없으면 협업 도구가 오히려 협업의 방해 요소가 될 수 있다.

핵심 098 소프트웨어 패키징의 개요

소프트웨어 패키징이란 모듈별로 생성한 실행 파일들을 묶어 배포용 설치 파일을 만드는 것을 말한다.

- 개발자가 아니라 사용자를 중심으로 진행한다.
- 소스 코드는 향후 관리를 고려하여 모듈화하여 패키징한다.
- 사용자가 소프트웨어를 사용하게 될 환경을 이해하여, 다양한 환경에서 소프트웨어를 손쉽게 사용할 수 있도록 일반적인 배포 형태로 패키징한다.

핵심 099 패키징 시 고려사항

- 사용자의 시스템 환경, 즉 운영체제(OS), CPU, 메모리 등에 필요한 최소 환경을 정의한다.
- UI(User Interface)는 사용자가 눈으로 직접 확인할 수 있도록 시각적인 자료와 함께 제공하고 매뉴얼과 일치시켜 패키징한다.
- 소프트웨어는 단순히 패키징하여 배포하는 것으로 끝나는 것이 아니라 하드웨어와 함께 관리될 수 있도록 Managed Service 형태로 제공하는 것이 좋다.
- 사용자에게 배포되는 소프트웨어이므로 내부 콘텐츠에 대한 암호화 및 보안을 고려한다.



- 다른 여러 콘텐츠 및 단말기 간 DRM(디지털 저작권 관리) 연동을 고려한다.
- 사용자의 편의성을 위한 복잡성 및 비효율성 문제를 고려한다.
- 제품 소프트웨어 종류에 적합한 암호화 알고리즘을 적용한다.

핵심 100 릴리즈 노트(Release Note)의 개요

릴리즈 노트는 개발 과정에서 정리된 릴리즈 정보를 소프트웨어의 최종 사용자인 고객과 공유하기 위한 문서이다.

- 릴리즈 노트를 통해 테스트 진행 방법에 대한 결과와 소프트웨어 사양에 대한 개발팀의 정확한 준수 여부를 확인할 수 있다.
- 소프트웨어에 포함된 전체 기능, 서비스의 내용, 개선 사항 등을 사용자와 공유할 수 있다.
- 릴리즈 노트를 이용해 소프트웨어의 버전 관리나 릴리즈 정보를 체계적으로 관리할 수 있다.
- 릴리즈 노트는 소프트웨어의 초기 배포 시 또는 출시 후 개선 사항을 적용한 추가 배포 시에 제공한다.

핵심 101 릴리즈 노트 초기 버전 작성 시 고려사항

릴리즈 노트의 초기 버전은 다음의 사항을 고려하여 작성한다.

- 릴리즈 노트는 정확하고 완전한 정보를 기반으로 개발 팀에서 직접 현재 시제로 작성해야 한다.
- 신규 소스, 빌드 등의 이력이 정확하게 관리되어 변경 또는 개선된 항목에 대한 이력 정보들도 작성되어야 한다.
- 릴리즈 노트 작성에 대한 표준 형식은 없지만 일반적으로 다음과 같은 항목이 포함된다.
 - Header(머릿말)

- 개요
- 목적
- 문제 요약
- 재현 항목
- 수정/개선 내용
- 사용자 영향도
- SW 지원 영향도
- 노트
- 면책 조항
- 연락처

핵심 102 릴리즈 노트 추가 버전 작성 시 고려사항

소프트웨어의 테스트 과정에서 베타 버전이 출시되거나 긴급한 버그 수정, 업그레이드와 같은 자체 기능 향상, 사용자 요청 등의 특수한 상황이 발생하는 경우 릴리즈 노트를 추가로 작성한다.

- 중대한 오류가 발생하여 긴급하게 수정하는 경우에는 릴리즈 버전을 출시하고 버그 번호를 포함한 모든 수정된 내용을 담아 릴리즈 노트를 작성한다.
- 소프트웨어에 대한 기능 업그레이드를 완료한 경우에는 릴리즈 버전을 출시하고 릴리즈 노트를 작성한다.
- 사용자로부터 접수된 요구사항에 의해 추가나 수정된 경우 자체 기능 향상과는 다른 별도의 릴리즈 버전으로 출시하고 릴리즈 노트를 작성한다.

핵심 103 저작권의 개요

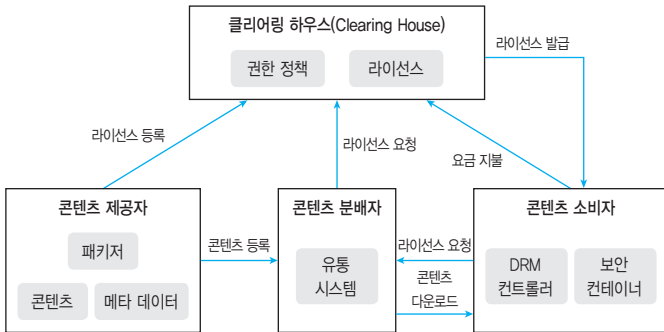
저작권이란 소설, 시, 논문, 강연, 연설, 음악, 연극, 무용, 회화, 서예, 건축물, 사진, 영상, 지도, 도표, 컴퓨터 프로그램 저작물 등에 대하여 창작자가 가지는 배타적 독점적 권리로 타인의 침해를 받지 않을 고유한 권한이다.

- 컴퓨터 프로그램들과 같이 복제하기 쉬운 저작물에 대해 불법 복제 및 배포 등을 막기 위한 기술적인 방법을 통칭해 저작권 보호 기술이라고 한다.

핵심 104 디지털 저작권 관리(DRM; Digital Right Management)

디지털 저작권 관리란 저작권자가 배포한 디지털 콘텐츠가 저작권자가 의도한 용도로만 사용되도록 디지털 콘텐츠의 생성, 유통, 이용까지의 전 과정에 걸쳐 사용되는 디지털 콘텐츠 관리 및 보호 기술이다.

디지털 저작권 관리의 흐름 및 구성 요소



클리어링 하우스 (Clearing House)	저작권에 대한 사용 권한, 라이선스 발급, 암호화된 키 관리, 사용량에 따른 결제 관리 등을 수행하는 곳
콘텐츠 제공자 (Contents Provider)	콘텐츠를 제공하는 저작권자
패키저(Packager)	콘텐츠를 메타 데이터와 함께 배포 가능한 형태로 묶어 암호화하는 프로그램
콘텐츠 분배자 (Contents Distributor)	암호화된 콘텐츠를 유통하는 곳이나 사람
콘텐츠 소비자 (Customer)	콘텐츠를 구매해서 사용하는 주체
DRM 컨트롤러 (DRM Controller)	배포된 콘텐츠의 이용 권한을 통제하는 프로그램
보안 컨테이너 (Security Container)	콘텐츠 원본을 안전하게 유통하기 위한 전자적 보안 장치

핵심 105 디지털 저작권 관리의 기술 요소

디지털 저작권 관리를 위해 사용되는 기술은 다음과 같다.

구성 요소	설명
암호화 (Encryption)	콘텐츠 및 라이선스를 암호화하고 전자 서명을 할 수 있는 기술
키 관리 (Key Management)	콘텐츠를 암호화한 키에 대한 저장 및 분배 기술
암호화 파일 생성 (Packager)	콘텐츠를 암호화된 콘텐츠로 생성하기 위한 기술
식별 기술 (Identification)	콘텐츠에 대한 식별 체계 표현 기술
저작권 표현 (Right Expression)	라이선스의 내용 표현 기술
정책 관리 (Policy Management)	라이선스 발급 및 사용에 대한 정책 표현 및 관리 기술
크랙 방지 (Tamper Resistance)	크랙에 의한 콘텐츠 사용 방지 기술
인증 (Authentication)	라이선스 발급 및 사용의 기준이 되는 사용자 인증 기술

핵심 106 소프트웨어 설치 매뉴얼의 개요

소프트웨어 설치 매뉴얼은 개발 초기에서부터 적용된 기준이나 사용자가 소프트웨어를 설치하는 과정에 필요한 내용을 기록한 설명서와 안내서이다.

- 설치 매뉴얼은 사용자 기준으로 작성한다.
- 설치 시작부터 완료할 때까지의 전 과정을 빠짐없이 순서대로 설명한다.
- 설치 과정에서 표시될 수 있는 오류 메시지 및 예외 상황에 관한 내용을 별도로 분류하여 설명한다.
- 소프트웨어 설치 매뉴얼에는 목차 및 개요, 서문, 기본 사항 등이 기본적으로 포함되어야 한다.

핵심 107 설치 매뉴얼 작성 방법

설치 매뉴얼은 사용자가 설치 과정을 이해하기 쉽도록 설치 화면을 누락 없이 캡처하고 순서대로 상세히 설명한다.

- 설치 매뉴얼에는 설치 화면 및 UI, 설치 이상 메시지, 설치 완료 및 결과, FAQ, 설치 시 점검 사항, Network 환경 및 보안, 고객 지원 방법, 준수 정보 및 제한 보증 등에 대한 내용을 기술한다.

설치 화면 및 UI	설치 실행과 메인 화면 및 안내창에 대한 내용을 기술한다.
설치 이상 메시지 설명	설치 방법이나 설치 환경이 잘못된 경우 표시될 수 있는 메시지에 대해 설명한다.
설치 완료 및 결과	설치 완료 화면을 수록하여 설치가 정상적으로 마무리되었음을 사용자에게 최종적으로 알려준다.
FAQ	설치 과정에서 사용자가 직면할 수 있는 문제 상황에 대비할 수 있도록, 설치 시 발생할 수 있는 다양한 상황을 FAQ로 정리하여 수록한다.
설치 시 점검 사항	설치 전 사용자의 설치 환경에 따라 점검해야 할 사항들이 무엇인지 설명한다.
Network 환경 및 보안	네트워크 오류로 인해 설치 시 문제가 발생하지 않도록 사전에 필요한 네트워크 연결 상태를 점검하도록 안내한다.
고객 지원 방법 (Customer Support)	설치와 관련하여 기술적인 지원이나 소프트웨어에 대한 서비스를 원할 경우 국가, 웹 사이트, 전화번호, 이메일 등 문의할 수 있는 연락처를 안내한다.
준수 정보 & 제한 보증 (Compliance Information & Limited Warranty)	Serial 보존, 불법 등록 사용 금지 등에 대한 준수 사항을 안내한다.

핵심 108 소프트웨어 사용자 매뉴얼의 개요

소프트웨어 사용자 매뉴얼은 사용자가 소프트웨어를 사용하는 과정에서 필요한 내용을 문서로 기록한 설명서와 안내서이다.

- 사용자 매뉴얼은 사용자가 소프트웨어 사용에 필요한 절차, 환경 등의 제반 사항이 모두 포함되도록 작성한다.
- 소프트웨어 배포 후 발생할 수 있는 오류에 대한 패치나 기능에 대한 업그레이드를 위해 매뉴얼의 버전을 관리한다.
- 개별적으로 동작이 가능한 컴포넌트 단위로 매뉴얼을 작성한다.
- 사용자 매뉴얼은 컴포넌트 명세서와 컴포넌트 구현 설계서를 토대로 작성한다.
- 사용자 매뉴얼에는 목차 및 개요, 서문, 기본 사항 등이 기본적으로 포함되어야 한다.

핵심 109 사용자 매뉴얼 작성 방법

사용자 매뉴얼은 사용자가 사용 방법을 이해하기 쉽도록 상황별로 누락 없이 캡처하여 순서대로 상세히 설명한다.

- 사용자 매뉴얼에는 사용자 화면 및 UI, 주요 기능 분류, 응용 프로그램 및 설정, 장치 연동, Network 환경, Profile 안내, 고객 지원 방법, 준수 정보 및 제한 보증 등에 대한 내용을 기술한다.

사용자 화면 및 UI	주의 사항과 참고 사항을 기술한다.
주요 기능 분류	기능이 실행되는 화면을 순서대로 캡처하여 기능에 대한 사용법을 설명한다.
응용 프로그램 (Program) 및 설정 (Setting)	소프트웨어 구동 시 함께 실행해도 되는 응용 프로그램, 또는 함께 실행되면 안 되는 응용 프로그램에 대해 설명한다.
장치 연동	소프트웨어가 특정 장치(Device)에 내장되는 경우 연동되는 장치(Device)에 대해 설명한다.
Network 환경	Network에 접속되어 사용되는 소프트웨어인 경우 정상적인 연결을 위한 설정값 등을 설명한다.

Profile 안내	Profile은 소프트웨어의 구동 환경을 점검하는 파일로, 사용자가 Profile의 경로를 변경하거나 위치를 이동하지 않도록 안내한다.
고객 지원 방법 (Customer Support)	사용과 관련하여 기술적인 지원이나 소프트웨어에 대한 서비스를 원할 경우 국가, 웹 사이트, 전화번호, 이메일 등 문의할 수 있는 연락처를 안내한다.
준수 정보 & 제한 보증(Compliance Information & Limited Warranty)	Serial 보존, 불법 등록 사용 금지 등에 대한 준수 사항을 안내한다.

핵심 110 소프트웨어 패키지의 형상 관리

형상 관리(SCM; Software Configuration Management)는 소프트웨어의 개발 과정에서 소프트웨어의 변경 사항을 관리하기 위해 개발된 일련의 활동이다.

- 소프트웨어 변경의 원인을 알아내고 제어하며, 적절히 변경되고 있는지 확인하여 해당 담당자에게 통보한다.
- 형상 관리는 소프트웨어 개발의 전 단계에 적용되는 활동이며, 유지보수 단계에서도 수행된다.
- 형상 관리는 소프트웨어 개발의 전체 비용을 줄이고, 개발 과정의 여러 방해 요인이 최소화되도록 보증하는 것을 목적으로 한다.
- 관리 항목에는 소스 코드뿐만 아니라 각종 정의서, 지침서, 분석서 등이 포함된다.

핵심 111 형상 관리의 중요성

- 지속적인 소프트웨어의 변경 사항을 체계적으로 추적하고 통제할 수 있다.
- 제품 소프트웨어에 대한 무절제한 변경을 방지할 수 있다.
- 제품 소프트웨어에서 발견된 버그나 수정 사항을 추적할 수 있다.
- 소프트웨어는 형태가 없어 가시성이 결핍되므로 진행 정도를 확인하기 위한 기준으로 사용될 수 있다.
- 소프트웨어의 배포본을 효율적으로 관리할 수 있다.
- 소프트웨어를 여러 명의 개발자가 동시에 개발할 수 있다.

핵심 112 소프트웨어의 버전 등록 관련 주요 기능

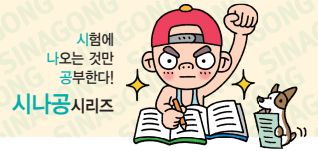
항목	설명
저장소 (Repository)	최신 버전의 파일들과 변경 내역에 대한 정보들이 저장되어 있는 곳이다.
가져오기 (Import)	버전 관리가 되고 있지 않은 아무것도 없는 저장소(Repository)에 처음으로 파일을 복사한다.
체크아웃 (Check-Out)	<ul style="list-style-type: none"> • 프로그램을 수정하기 위해 저장소(Repository)에서 파일을 받아온다. • 소스 파일과 함께 버전 관리를 위한 파일들도 받아온다.
체크인 (Check-In)	체크아웃 한 파일의 수정을 완료한 후 저장소(Repository)의 파일을 새로운 버전으로 갱신한다.
커밋 (Commit)	체크인을 수행할 때 이전에 갱신된 내용이 있는 경우에는 충돌(Conflict)을 알리고 diff 도구를 이용해 수정한 후 갱신을 완료한다.
동기화 (Update)	저장소에 있는 최신 버전으로 자신의 작업 공간을 동기화한다.

핵심 113 공유 폴더 방식 / 클라이언트/서버 방식 / 분산 저장소 방식

공유 폴더 방식

공유 폴더 방식은 버전 관리 자료가 로컬 컴퓨터의 공유 폴더에 저장되어 관리되는 방식으로, 다음과 같은 특징이 있다.

- 개발자들은 개발이 완료된 파일을 약속된 공유 폴더에 매일 복사한다.
- 담당자는 공유 폴더의 파일을 자기 PC로 복사한 후 컴파일 하여 이상 유무를 확인한다.
- 종류에는 SCCS, RCS, PVCS, QVCS 등이 있다.



클라이언트/서버 방식

클라이언트/서버 방식은 버전 관리 자료가 중앙 시스템 (서버)에 저장되어 관리되는 방식으로, 다음과 같은 특징이 있다.

- 서버의 자료를 개발자별로 자신의 PC(클라이언트)로 복사하여 작업한 후 변경된 내용을 서버에 반영한다.
- 모든 버전 관리는 서버에서 수행된다.
- 종류에는 CVS, SVN(Subversion), CVSNT, Clear Case, CMVC, Perforce 등이 있다.

분산 저장소 방식

분산 저장소 방식은 버전 관리 자료가 하나의 원격 저장소와 분산된 개발자 PC의 로컬 저장소에 함께 저장되어 관리되는 방식으로, 다음과 같은 특징이 있다.

- 개발자별로 원격 저장소의 자료를 자신의 로컬 저장소로 복사하여 작업한 후 변경된 내용을 로컬 저장소에서 우선 반영(버전 관리)한 다음 이를 원격 저장소에 반영한다.
- 종류에는 Git, GNU arch, DVC, Bazaar, Mercurial, TeamWare, Bitkeeper, Plastic SCM 등이 있다.

핵심 114 Subversion(서브버전, SVN)

Subversion은 CVS를 개선한 것으로, 아파치 소프트웨어 재단에서 2000년에 발표하였다.

- 클라이언트/서버 구조로, 서버(저장소, Repository)에는 최신 버전의 파일들과 변경 내역이 관리된다.
- 서버의 자료를 클라이언트로 복사해와 작업한 후 변경 내용을 서버에 반영(Commit)한다.
- 모든 개발 작업은 trunk 디렉터리에서 수행되며, 추가 작업은 branches 디렉터리 안에 별도의 디렉터를 만들어 작업을 완료한 후 trunk 디렉터리와 병합(merge)한다.
- 커밋(Commit)할 때마다 리비전(Revision)이 1씩 증가한다.

- 클라이언트는 대부분의 운영체제에서 사용되지만, 서버는 주로 유닉스를 사용한다.
- 소스가 오픈되어 있어 무료로 사용할 수 있다.
- CVS의 단점이었던 파일이나 디렉터리의 이름 변경, 이동 등이 가능하다.

핵심 115 Subversion의 주요 명령어

명령어	의미
add	<ul style="list-style-type: none"> • 새로운 파일이나 디렉터를 버전 관리 대상으로 등록한다. • add로 등록되지 않은 대상은 commit이 적용되지 않는다.
commit	버전 관리 대상으로 등록된 클라이언트의 소스 파일을 서버의 소스 파일에 적용한다.
update	<ul style="list-style-type: none"> • 서버의 최신 commit 이력을 클라이언트의 소스 파일에 적용한다. • commit 전에는 매번 update를 수행하여 클라이언트에 적용되지 않은 서버의 변동 내역을 클라이언트에 적용한다.
checkout	버전 관리 정보와 소스 파일을 서버에서 클라이언트로 받아온다.
lock/unlock	서버의 소스 파일이나 디렉터를 잠그거나 해제한다.
import	아무것도 없는 서버의 저장소에 맨 처음 소스 파일을 저장하는 명령으로, 한 번 사용하면 다시 사용하지 않는다.
export	버전 관리에 대한 정보를 제외한 순수한 소스 파일만을 서버에서 받아온다.
info	지정한 파일에 대한 위치나 마지막 수정 일자 등에 대한 정보를 표시한다.
diff	지정된 파일이나 경로에 대해 이전 리비전과의 차이를 표시한다.
merge	다른 디렉터리에서 작업된 버전 관리 내역을 기본 개발 작업과 병합한다.

핵심 116 Git(깃)

Git은 리누스 토발즈(Linus Torvalds)가 2005년 리눅스 커널 개발에 사용할 관리 도구로 개발한 이후 주니오 하마노(Junio Hamano)에 의해 유지 보수되고 있다.

- Git은 분산 버전 관리 시스템으로 2개의 저장소, 즉 지역(로컬) 저장소와 원격 저장소가 존재한다.
- 지역 저장소는 개발자들이 실제 개발을 진행하는 장소로, 버전 관리가 수행된다.
- 원격 저장소는 여러 사람들이 협업을 위해 버전을 공동 관리하는 곳으로, 자신의 버전 관리 내역을 반영하거나 다른 개발자의 변경 내용을 가져올 때 사용한다.
- 버전 관리가 지역 저장소에서 진행되므로 버전 관리가 신속하게 처리되고, 원격 저장소나 네트워크에 문제가 있어도 작업이 가능하다.
- 브랜치를 이용하면 기본 버전 관리 틀에 영향을 주지 않으면서 다양한 형태의 기능 테스트가 가능하다.
- 파일의 변화를 스냅샷(Snapshot)으로 저장하는데, 스냅샷은 이전 스냅샷의 포인터를 가지므로 버전의 흐름을 파악할 수 있다.

checkout	<ul style="list-style-type: none"> • 지정한 브랜치로 이동한다. • 현재 작업 중인 브랜치는 HEAD 포인터가 가리키는데, checkout 명령을 통해 HEAD 포인터를 지정한 브랜치로 이동시킨다.
merge	지정한 브랜치의 변경 내역을 현재 HEAD 포인터가 가리키는 브랜치에 반영함으로써 두 브랜치를 병합한다.
init	지역 저장소를 생성한다.
remote add	원격 저장소에 연결한다.
push	로컬 저장소의 변경 내역을 원격 저장소에 반영한다.
fetch	원격 저장소의 변경 이력만을 지역 저장소로 가져와 반영한다.
clone	원격 저장소의 전체 내용을 지역 저장소로 복제한다.
fork	지정한 원격 저장소의 내용을 자신의 원격 저장소로 복제한다.

핵심 117 Git의 주요 명령어

명령어	의미
add	<ul style="list-style-type: none"> • 작업 내역을 지역 저장소에 저장하기 위해 스테이징 영역(Staging Area)에 추가한다. • '-a' 옵션으로 작업 디렉터리의 모든 파일을 스테이징 영역에 추가할 수 있다.
commit	작업 내역을 지역 저장소에 저장한다.
branch	<ul style="list-style-type: none"> • 새로운 브랜치를 생성한다. • 최초로 commit을 하면 마스터(master) 브랜치가 생성된다. • commit 할 때마다 해당 브랜치는 가장 최근의 commit한 내용을 가리키게 된다. • '-d' 옵션으로 브랜치를 삭제할 수 있다.

핵심 118 빌드 자동화 도구의 개념

빌드란 소스 코드 파일들을 컴파일한 후 여러 개의 모듈을 묶어 실행 파일로 만드는 과정이며, 이러한 빌드를 포함하여 테스트 및 배포를 자동화하는 도구를 빌드 자동화 도구라고 한다.

- 애자일 환경에서는 하나의 작업이 마무리될 때마다 모듈 단위로 나눠서 개발된 코드들이 지속적으로 통합되는데, 이러한 지속적인 통합(Continuous Integration) 개발 환경에서 빌드 자동화 도구는 유용하게 활용된다.
- 빌드 자동화 도구에는 Ant, Make, Maven, Gradle, Jenkins 등이 있으며, 이중 Jenkins와 Gradle이 가장 대표적이다.

핵심 119 Jenkins/Gradle

Jenkins

Jenkins는 JAVA 기반의 오픈 소스 형태로, 가장 많이 사용되는 빌드 자동화 도구이다.

- 서블릿 컨테이너에서 실행되는 서버 기반 도구이다.
- SVN, Git 등 대부분의 형상 관리 도구와 연동이 가능하다.
- 친숙한 Web GUI 제공으로 사용이 쉽다.
- 여러 대의 컴퓨터를 이용한 분산 빌드나 테스트가 가능하다.

Gradle

Gradle은 Groovy를 기반으로 한 오픈 소스 형태의 자동화 도구로, 안드로이드 앱 개발 환경에서 사용된다.

- 안드로이드 뿐만 아니라 플러그인을 설정하면, JAVA, C/C++, Python 등의 언어도 빌드가 가능하다.
- Groovy를 사용해서 만든 DSL(Domain Specific Language)을 스크립트 언어로 사용한다.
- Gradle은 실행할 처리 명령들을 모아 태스크(Task)로 만든 후 태스크 단위로 실행한다.
- 이전에 사용했던 태스크를 재사용하거나 다른 시스템의 태스크를 공유할 수 있는 빌드 캐시 기능을 지원하므로 빌드의 속도를 향상시킬 수 있다.

핵심 120 애플리케이션 테스트

애플리케이션 테스트의 개념

- 애플리케이션 테스트는 애플리케이션에 잠재되어 있는 결함을 찾아내는 일련의 행위 또는 절차이다.
- 애플리케이션 테스트는 개발된 소프트웨어가 고객의 요구사항을 만족시키는지 확인(Validation)하고 소프트웨어가 기능을 정확히 수행하는지 검증(Verification)한다.

애플리케이션 테스트의 기본 원리

- 애플리케이션 테스트는 소프트웨어의 잠재적인 결함을 줄일 수 있지만 소프트웨어에 결함이 없다고 증명할 수는 없다. 즉 완벽한 소프트웨어 테스트는 불가능하다.

- 애플리케이션의 결함은 대부분 개발자의 특성이나 애플리케이션의 기능적 특징 때문에 특정 모듈에 집중되어 있다. 애플리케이션의 20%에 해당하는 코드에서 전체 80%의 결함이 발견된다고 하여 파레토 법칙을 적용하기도 한다.
- 애플리케이션 테스트에서는 동일한 테스트 케이스로 동일한 테스트를 반복하면 더 이상 결함이 발견되지 않는 '살충제 패러독스(Pesticide Paradox)' 현상이 발생한다. 살충제 패러독스를 방지하기 위해서 테스트 케이스를 지속적으로 보완 및 개선해야 한다.
- 애플리케이션 테스트는 소프트웨어 특징, 테스트 환경, 테스트 역량 등 정황(Context)에 따라 테스트 결과가 달라질 수 있으므로, 정황에 따라 테스트를 다르게 수행해야 한다.
- 소프트웨어의 결함을 모두 제거해도 사용자의 요구사항을 만족시키지 못하면 해당 소프트웨어는 품질이 높다고 말할 수 없다. 이것을 오류-부재의 궤변(Absence of Errors Fallacy)이라고 한다.
- 테스트와 위험은 반비례한다. 테스트를 많이 하면 할수록 미래에 발생할 위험을 줄일 수 있다.
- 테스트는 작은 부분에서 시작하여 점점 확대하며 진행해야 한다.
- 테스트는 개발자와 관계없는 별도의 팀에서 수행해야 한다.

핵심 121 애플리케이션 테스트의 분류

프로그램 실행 여부에 따른 테스트

정적 테스트	<ul style="list-style-type: none"> • 프로그램을 실행하지 않고 명세나 소스 코드를 대상으로 분석하는 테스트이다. • 소프트웨어 개발 초기에 결함을 발견할 수 있어 소프트웨어의 개발 비용을 낮추는데 도움이 된다. • 종류 : 워크루, 인스펙션, 코드 검사 등
동적 테스트	<ul style="list-style-type: none"> • 프로그램을 실행하여 오류를 찾는 테스트로, 소프트웨어 개발의 모든 단계에서 테스트를 수행할 수 있다. • 종류 : 블랙박스 테스트, 화이트박스 테스트

테스트 기반(Test Bases)에 따른 테스트

명세 기반 테스트	<ul style="list-style-type: none"> • 사용자의 요구사항에 대한 명세를 빠짐없이 테스트 케이스로 만들어 구현하고 있는지 확인하는 테스트이다. • 종류 : 동등 분할, 경계 값 분석 등
구조 기반 테스트	<ul style="list-style-type: none"> • 소프트웨어 내부의 논리 흐름에 따라 테스트 케이스를 작성하고 확인하는 테스트이다. • 종류 : 구문 기반, 결정 기반, 조건 기반 등
경험 기반 테스트	<ul style="list-style-type: none"> • 유사 소프트웨어나 기술 등에 대한 테스트의 경험을 기반으로 수행하는 테스트이다. • 경험 기반 테스트는 사용자의 요구사항에 대한 명세가 불충분하거나 테스트 시간에 제약이 있는 경우 수행하면 효과적이다. • 종류 : 에러 추정, 체크 리스트, 탐색적 테스트

시각에 따른 테스트

검증(Verification) 테스트	개발자의 시각에서 제품의 생산 과정을 테스트하는 것으로, 제품이 명세서대로 완성됐는지를 테스트한다.
확인(Validation) 테스트	사용자의 시각에서 생산된 제품의 결과를 테스트하는 것으로, 사용자가 요구한대로 제품이 완성됐는지, 제품이 정상적으로 동작하는지를 테스트한다.

목적에 따른 테스트

회복(Recovery) 테스트	시스템에 여러 가지 결함을 주어 실패하도록 한 후 올바르게 복구되는지를 확인하는 테스트이다.
안전(Security) 테스트	시스템에 설치된 시스템 보호 도구가 불법적인 침입으로부터 시스템을 보호할 수 있는지를 확인하는 테스트이다.
강도(Stress) 테스트	시스템에 과도한 정보량이나 빈도 등을 부과하여 과부하 시에도 소프트웨어가 정상적으로 실행되는지를 확인하는 테스트이다.
성능(Performance) 테스트	소프트웨어의 실시간 성능이나 전체적인 효율성을 진단하는 테스트로, 소프트웨어의 응답 시간, 처리량 등을 테스트한다.
구조(Structure) 테스트	소프트웨어 내부의 논리적인 경로, 소스 코드의 복잡도 등을 평가하는 테스트이다.
회귀(Regression) 테스트	소프트웨어의 변경 또는 수정된 코드에 새로운 결함이 없음을 확인하는 테스트이다.
병행(Parallel) 테스트	변경된 소프트웨어와 기존 소프트웨어에 동일한 데이터를 입력하여 결과를 비교하는 테스트이다.

핵심 122 화이트박스 테스트(White Box Test)

화이트박스 테스트는 모듈의 원시 코드를 오픈시킨 상태에서 원시 코드의 논리적인 모든 경로를 테스트하여 테스트 케이스를 설계하는 방법이다.

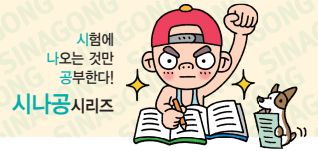
- 모듈 안의 작동을 직접 관찰한다.
- 원시 코드(모듈)의 모든 문장을 한 번 이상 실행함으로써 수행된다.

화이트박스 테스트의 종류

기초 경로 검사 (Base Path Testing)	<ul style="list-style-type: none"> • 대표적인 화이트박스 테스트 기법이다. • 테스트 케이스 설계자가 절차적 설계의 논리적 복잡성을 측정할 수 있게 해주는 테스트 기법으로, 테스트 측정 결과는 실행 경로의 기초를 정의하는 데 지침으로 사용된다.
제어 구조 검사 (Control Structure Testing)	<ul style="list-style-type: none"> • 조건 검사(Condition Testing) : 프로그램 모듈 내에 있는 논리적 조건을 테스트하는 테스트 케이스 설계 기법 • 루프 검사(Loop Testing) : 프로그램의 반복(Loop) 구조에 초점을 맞춰 실시하는 테스트 케이스 설계 기법 • 데이터 흐름 검사(Data Flow Testing) : 프로그램에서 변수의 정의와 변수 사용의 위치에 초점을 맞춰 실시하는 테스트 케이스 설계 기법

화이트박스 테스트의 검증 기준

문장 검증 기준 (Statement Coverage)	소스 코드의 모든 구문이 한 번 이상 수행되도록 테스트 케이스 설계
분기 검증 기준 (Branch Coverage)	소스 코드의 모든 조건문이 한 번 이상 수행되도록 테스트 케이스 설계
조건 검증 기준 (Condition Coverage)	소스 코드의 모든 조건문에 대해 조건이 True인 경우와 False인 경우가 한 번 이상 수행되도록 테스트 케이스 설계
분기/조건 기준 (Branch/Condition Coverage)	소스 코드의 모든 조건문과 각 조건문에 포함된 개별 조건식의 결과가 True인 경우와 False인 경우가 한 번 이상 수행되도록 테스트 케이스 설계



핵심 123 블랙박스 테스트(Black Box Test)

블랙박스 테스트는 소프트웨어가 수행할 특정 기능을 알기 위해서 각 기능이 완전히 작동되는 것을 입증하는 테스트로, 기능 테스트라고도 한다.

- 사용자의 요구사항 명세를 보면서 테스트하는 것으로, 주로 구현된 기능을 테스트한다.
- 소프트웨어 인터페이스에서 실시되는 테스트이다.

블랙박스 테스트의 종류

동치 분할 검사 (Equivalence Partitioning Testing, 동치 클래스 분해)	<ul style="list-style-type: none"> • 입력 자료에 초점을 맞춰 테스트 케이스(동치 클래스)를 만들고 검사하는 방법으로 동치 분할 기법이라고도 한다. • 프로그램의 입력 조건에 타당한 입력 자료와 타당하지 않은 입력 자료의 개수를 균등하게 하여 테스트 케이스를 정하고, 해당 입력 자료에 맞는 결과가 출력되는지 확인하는 기법이다.
경계값 분석 (Boundary Value Analysis)	<ul style="list-style-type: none"> • 입력 자료에만 치중한 동치 분할 기법을 보완하기 위한 기법이다. • 입력 조건의 중간값보다 경계값에서 오류가 발생될 확률이 높다는 점을 이용하여 입력 조건의 경계값을 테스트 케이스로 선정하여 검사하는 기법이다.
원인-효과 그래프 검사 (Cause-Effect Graphing Testing)	입력 데이터 간의 관계와 출력에 영향을 미치는 상황을 체계적으로 분석한 다음 효용성이 높은 테스트 케이스를 선정하여 검사하는 기법이다.
오류 예측 검사 (Error Guessing)	<ul style="list-style-type: none"> • 과거의 경험이나 확인자의 감각으로 테스트하는 기법이다. • 다른 블랙 박스 테스트 기법으로는 찾아낼 수 없는 오류를 찾아내는 일련의 보충적 검사 기법이며, 데이터 확인 검사라고도 한다.
비교 검사 (Comparison Testing)	여러 버전의 프로그램에 동일한 테스트 자료를 제공하여 동일한 결과가 출력되는지 테스트하는 기법이다.

핵심 124 개발 단계에 따른 애플리케이션 테스트

단위 테스트 (Unit Test)	<ul style="list-style-type: none"> • 코딩 직후 소프트웨어 설계의 최소 단위인 모듈이나 컴포넌트에 초점을 맞춰 테스트하는 것이다. • 인터페이스, 외부적 I/O, 자료 구조, 독립적 기초 경로, 오류 처리 경로, 경계 조건 등을 검사한다.
통합 테스트 (Integration Test)	<ul style="list-style-type: none"> • 단위 테스트가 완료된 모듈들을 결합하여 하나의 시스템으로 완성시키는 과정에서의 테스트를 의미한다. • 모듈 간 또는 통합된 컴포넌트 간의 상호 작용 오류를 검사한다.
시스템 테스트 (System Test)	<ul style="list-style-type: none"> • 개발된 소프트웨어가 해당 컴퓨터 시스템에서 완벽하게 수행되는가를 점검하는 테스트이다. • 기능적 요구사항과 비기능적 요구사항으로 구분하여 각각을 만족하는지 테스트한다.
인수 테스트 (Acceptance Test)	<ul style="list-style-type: none"> • 개발한 소프트웨어가 사용자의 요구사항을 충족하는지에 중점을 두고 테스트하는 방법이다. • 개발한 소프트웨어를 사용자가 직접 테스트한다.

• 인수 테스트 종류

사용자 인수 테스트	사용자가 시스템 사용의 적절성 여부를 확인한다.
운영상의 인수 테스트	시스템 관리자가 시스템 인수 시 수행하는 테스트 기법으로, 백업/복원 시스템, 재난 복구, 사용자 관리, 정기 점검 등을 확인한다.
계약 인수 테스트	계약상의 인수/검수 조건을 준수하는지 여부를 확인한다.
규정 인수 테스트	소프트웨어가 정부 지침, 법규, 규정 등 규정에 맞게 개발되었는지 확인한다.
알파 테스트	<ul style="list-style-type: none"> • 개발자의 장소에서 사용자가 개발자 앞에서 행하는 테스트 기법이다. • 테스트는 통제된 환경에서 행해지며, 오류와 사용상의 문제점을 사용자와 개발자가 함께 확인하면서 기록한다.
베타 테스트	<ul style="list-style-type: none"> • 선정된 최종 사용자가 여러 명의 사용자 앞에서 행하는 테스트 기법이다. • 실업무를 가지고 사용자가 직접 테스트하는 것으로, 개발자에 의해 제어되지 않은 상태에서 테스트가 행해지며, 발견된 오류와 사용상의 문제점을 기록하고 개발자에게 주기적으로 보고한다.

핵심

125

하향식 통합 테스트 (Top Down Integration Test)

하향식 통합 테스트는 프로그램의 상위 모듈에서 하위 모듈 방향으로 통합하면서 테스트하는 기법이다.

- 주요 제어 모듈을 기준으로 하여 아래 단계로 이동하면서 통합하는데, 이때 깊이 우선 통합법이나 넓이 우선 통합법을 사용한다.
- 테스트 초기부터 사용자에게 시스템 구조를 보여줄 수 있다.
- 상위 모듈에서는 테스트 케이스를 사용하기 어렵다.
- 하향식 통합 방법은 다음과 같은 절차로 수행된다.

- ① 주요 제어 모듈은 작성된 프로그램을 사용하고, 주요 제어 모듈의 종속 모듈들은 스텝(Stub)으로 대체한다.

※ 테스트 스텝(Test Stub) : 제어 모듈이 호출하는 타 모듈의 기능을 단순히 수행하는 도구로, 일시적으로 필요한 조건만을 가지고 있는 시험용 모듈이다.

- ② 깊이 우선 또는 넓이 우선 등의 통합 방식에 따라 하위 모듈인 스텝들이 한 번에 하나씩 실제 모듈로 교체된다.
- ③ 모듈이 통합될 때마다 테스트를 실시한다.
- ④ 새로운 오류가 발생하지 않음을 보증하기 위해 회귀 테스트를 실시한다.

핵심

126

상향식 통합 테스트 (Bottom Up Integration Test)

상향식 통합 테스트는 프로그램의 하위 모듈에서 상위 모듈 방향으로 통합하면서 테스트하는 기법이다.

- 가장 하위 단계의 모듈부터 통합 및 테스트가 수행되므로 스텝(Stub)은 필요하지 않지만, 하나의 주요 제어 모듈과 관련된 종속 모듈의 그룹인 클러스터(Cluster)가 필요하다.

- 상향식 통합 방법은 다음과 같은 절차로 수행된다.

- ① 하위 모듈들을 클러스터(Cluster)로 결합한다.
 - ② 상위 모듈에서 데이터의 입·출력을 확인하기 위해 더미 모듈인 드라이버(Driver)를 작성한다.
- ※ 테스트 드라이버(Test Driver) : 테스트 드라이버는 테스트 대상의 하위 모듈을 호출하고, 파라미터를 전달하고, 모듈 테스트 수행 후의 결과를 도출하는 도구이다.
- ③ 통합된 클러스터 단위로 테스트한다.
 - ④ 테스트가 완료되면 클러스터는 프로그램 구조의 상위로 이동하여 결합하고 드라이버는 실제 모듈로 대체된다.

핵심

127

애플리케이션 테스트 프로세스

- 애플리케이션 테스트 프로세스는 개발된 소프트웨어가 사용자의 요구대로 만들어졌는지, 결함은 없는지 등을 테스트하는 절차이다.

- 순서

테스트 계획	프로젝트 계획서, 요구 명세서 등을 기반으로 테스트 목표를 정의하고 테스트 대상 및 범위를 결정한다.
테스트 분석 및 디자인	테스트의 목적과 원칙을 검토하고 사용자의 요구 사항을 분석한다.
테스트 케이스 및 시나리오 작성	테스트 케이스의 설계 기법에 따라 테스트 케이스를 작성하고 검토 및 확인한 후 테스트 시나리오를 작성한다.
테스트 수행	<ul style="list-style-type: none"> • 테스트 환경을 구축한 후 테스트를 수행한다. • 테스트의 실행 결과를 측정하여 기록한다.
테스트 결과 평가 및 리포트	테스트 결과를 비교 분석하여 테스트 결과서를 작성한다.
결함 추적 및 관리	테스트를 수행한 후 결함이 어디에서 발생했는지, 어떤 종류의 결함인지 등 결함을 추적하고 관리한다.

• 결함 관리 프로세스

- ① 에러 발견 : 에러가 발견되면 테스트 전문가와 프로젝트팀이 논의한다.
- ② 에러 등록 : 발견된 에러를 결함 관리 대장에 등록한다.
- ③ 에러 분석 : 등록된 에러가 실제 결함인지 아닌지를 분석한다.
- ④ 결함 확정 : 등록된 에러가 실제 결함이면 결함 확정 상태로 설정한다.
- ⑤ 결함 할당 : 결함을 해결할 담당자에게 결함을 할당하고 결함 할당 상태로 설정한다.
- ⑥ 결함 조치 : 결함을 수정하고, 수정이 완료되면 결함 조치 상태로 설정한다.
- ⑦ 결함 조치 검토 및 승인 : 수정이 완료된 결함에 대해 확인 테스트를 수행하고, 이상이 없으면 결함 조치 완료 상태로 설정한다.

핵심

128

테스트 케이스 / 테스트 시나리오 / 테스트 오라클

테스트 케이스(Test Case)

- 테스트 케이스는 구현된 소프트웨어가 사용자의 요구 사항을 정확하게 준수했는지를 확인하기 위해 설계된 입력 값, 실행 조건, 기대 결과 등으로 구성된 테스트 항목에 대한 명세서로, 명세 기반 테스트의 설계 산출물에 해당된다.
- 테스트 케이스를 미리 설계하면 테스트 오류를 방지할 수 있고 테스트 수행에 필요한 인력, 시간 등의 낭비를 줄일 수 있다.

테스트 시나리오(Test Scenario)

- 테스트 시나리오는 테스트 케이스를 적용하는 순서에 따라 여러 개의 테스트 케이스들을 묶은 집합으로, 테스트 케이스들을 적용하는 구체적인 절차를 명세한 문서이다.
- 테스트 시나리오에는 테스트 순서에 대한 구체적인 절차, 사전 조건, 입력 데이터 등이 설정되어 있다.

테스트 오라클(Test Oracle)

- 테스트 오라클은 테스트 결과가 올바른지 판단하기 위해 사전에 정의된 참 값을 대입하여 비교하는 기법 및 활동을 말한다.
- 테스트 오라클은 결과를 판단하기 위해 테스트 케이스에 대한 예상 결과를 계산하거나 확인한다.
- 테스트 오라클의 특징
 - 제한된 검증 : 테스트 오라클을 모든 테스트 케이스에 적용할 수 없다.
 - 수학적 기법 : 테스트 오라클의 값을 수학적 기법을 이용하여 구할 수 있다.
 - 자동화 기능 : 테스트 대상 프로그램의 실행, 결과 비교, 커버리지 측정 등을 자동화 할 수 있다.

테스트 오라클의 종류

참(True) 오라클	모든 테스트 케이스의 입력 값에 대해 기대하는 결과를 제공하는 오라클로, 발생된 모든 오류를 검출할 수 있다.
샘플링(Sampling) 오라클	특정한 몇몇 테스트 케이스의 입력 값들에 대해서만 기대하는 결과를 제공하는 오라클이다.
추정(Heuristic) 오라클	샘플링 오라클을 개선한 오라클로, 특정 테스트 케이스의 입력 값에 대해 기대하는 결과를 제공하고, 나머지 입력 값들에 대해서는 추정으로 처리하는 오라클이다.
일관성 검사(Consistent) 오라클	애플리케이션의 변경이 있을 때, 테스트 케이스의 수행 전과 후의 결과 값이 동일한지를 확인하는 오라클이다.

핵심

129

테스트 자동화 도구

테스트 자동화의 개념

테스트 자동화는 사람이 반복적으로 수행하던 테스트 절차를 스크립트 형태로 구현하는 자동화 도구를 적용함으로써 쉽고 효율적으로 테스트를 수행할 수 있도록 한 것이다.

테스트 자동화 도구의 유형

정적 분석 도구 (Static Analysis Tools)	프로그램을 실행하지 않고 분석하는 도구로, 소스 코드에 대한 코딩 표준, 코딩 스타일, 코드 복잡도 및 남은 결함 등을 발견하기 위해 사용된다.
테스트 실행 도구 (Test Execution Tools)	<ul style="list-style-type: none"> 스크립트 언어를 사용하여 테스트를 실행하는 방법으로, 테스트 데이터와 테스트 수행 방법 등이 포함된 스크립트를 작성한 후 실행한다. 데이터 주도 접근 방식 : 스프레드시트에 테스트 데이터를 저장하고, 이를 읽어 실행하는 방식이다. 키워드 주도 접근 방식 : 스프레드시트에 테스트를 수행할 동작을 나타내는 키워드와 테스트 데이터를 저장하여 실행하는 방식이다.
성능 테스트 도구 (Performance Test Tools)	애플리케이션의 처리량, 응답 시간, 경과 시간, 자원 사용률 등을 인위적으로 적용한 가상의 사용자를 만들어 테스트를 수행함으로써 성능의 목표 달성 여부를 확인한다.
테스트 통제 도구 (Test Control Tools)	테스트 계획 및 관리, 테스트 수행, 결함 관리 등을 수행하는 도구로, 종류에는 형상 관리 도구, 결함 추적/관리 도구 등이 있다.
테스트 하네스 도구 (Test Harness Tools)	<ul style="list-style-type: none"> 테스트 하네스는 애플리케이션의 컴포넌트 및 모듈을 테스트하는 환경의 일부분으로, 테스트를 지원하기 위해 생성된 코드와 데이터를 의미한다. 테스트 하네스 도구는 테스트가 실행될 환경을 시뮬레이션 하여 컴포넌트 및 모듈이 정상적으로 테스트되도록 한다.

※ 테스트 하네스(Test Harness)의 구성 요소

- 테스트 드라이버(Test Driver) : 테스트 대상의 하위 모듈을 호출하고, 파라미터를 전달하고, 모듈 테스트 수행 후의 결과를 도출하는 도구
- 테스트 스텝(Test Stub) : 제어 모듈이 호출하는 타 모듈의 기능을 단순히 수행하는 도구로, 일시적으로 필요한 조건만을 가지고 있는 테스트용 모듈
- 테스트 슈트(Test Suites) : 테스트 대상 컴포넌트나 모듈, 시스템에 사용되는 테스트 케이스의 집합
- 테스트 케이스(Test Case) : 사용자의 요구사항을 정확하게 준수했는지 확인하기 위한 입력 값, 실행 조건, 기대 결과 등으로 만들어진 테스트 항목의 명세서
- 테스트 스크립트(Test Script) : 자동화된 테스트 실행 절차에 대한 명세서
- Mock 오브젝트(Mock Object) : 사전에 사용자의 행위를 조건부로 입력해 두면, 그 상황에 맞는 예정된 행위를 수행하는 객체

테스트 수행 단계별 테스트 자동화 도구

테스트 단계	자동화 도구	설명
테스트 계획	요구사항 관리	사용자의 요구사항 정의 및 변경 사항 등을 관리하는 도구
테스트 분석/설계	테스트 케이스 생성	테스트 기법에 따른 테스트 데이터 및 테스트 케이스 작성을 지원하는 도구
테스트 수행	테스트 자동화	테스트의 자동화를 도와주는 도구로 테스트의 효율성을 높임
	정적 분석	코딩 표준, 런타임 오류 등을 검증하는 도구
	동적 분석	대상 시스템의 시뮬레이션을 통해 오류를 검출하는 도구
	성능 테스트	가상의 사용자를 생성하여 시스템의 처리 능력을 측정하는 도구
	모니터링	CPU, Memory 등과 같은 시스템 자원의 상태 확인 및 분석을 지원하는 도구
테스트 관리	커버리지 분석	테스트 완료 후 테스트의 충분성 여부 검증을 지원하는 도구
	형상 관리	테스트 수행에 필요한 다양한 도구 및 데이터를 관리하는 도구
	결함 추적/관리	테스트 시 발생한 결함 추적 및 관리 활동을 지원하는 도구

핵심 130 결함 관리

결함(Fault)의 정의

결함은 오류 발생, 작동 실패 등과 같이 소프트웨어가 개발자가 설계한 것과 다르게 동작하거나 다른 결과가 발생하는 것을 의미한다.

- 사용자가 예상한 결과와 실행 결과 간의 차이나 업무 내용과의 불일치 등으로 인해 변경이 필요한 부분도 모두 결함에 해당된다.

결함 상태 추적

테스트에서 발견된 결함은 지속적으로 상태 변화를 추적하고 관리해야 한다.

- 발견된 결함에 대해 결함 관리 측정 지표의 속성 값들을 분석하여 향후 결함이 발견될 모듈 또는 컴포넌트를 추정할 수 있다.

• 결함 관리 측정 지표

결함 분포	모듈 또는 컴포넌트의 특정 속성에 해당하는 결함 수 측정
결함 추세	테스트 진행 시간에 따른 결함 수의 추이 분석
결함 에이징	특정 결함 상태로 지속되는 시간 측정

결함 추적 순서

결함 추적은 결함이 발견된 때부터 결함이 해결될 때까지 전 과정을 추적하는 것으로 순서는 다음과 같다.

- ① 결함 등록(Open) : 테스터와 품질 관리(QA) 담당자에 의해 발견된 결함이 등록된 상태
- ② 결함 검토(Reviewed) : 등록된 결함을 테스터, 품질 관리(QA) 담당자, 프로그램 리더, 담당 모듈 개발자에 의해 검토된 상태
- ③ 결함 할당(Assigned) : 결함을 수정하기 위해 개발자와 문제 해결 담당자에게 결함이 할당된 상태
- ④ 결함 수정(Resolved) : 개발자가 결함 수정을 완료한 상태
- ⑤ 결함 조치 보류(Deferred) : 결함의 수정이 불가능해 연기된 상태로, 우선순위, 일정 등에 따라 재오픈을 준비중인 상태
- ⑥ 결함 종료(Closed) : 결함이 해결되어 테스터와 품질 관리(QA) 담당자가 종료를 승인한 상태
- ⑦ 결함 해제(Clarified) : 테스터, 프로그램 리더, 품질 관리(QA) 담당자가 종료 승인한 결함을 검토하여 결함이 아니라고 판명한 상태

결함 분류

시스템 결함	시스템 다운, 애플리케이션의 작동 정지, 종료, 응답 시간 지연, 데이터베이스 에러 등 주로 애플리케이션 환경이나 데이터베이스 처리에서 발생된 결함
기능 결함	사용자의 요구사항 미반영/불일치, 부정확한 비즈니스 프로세스, 스크립트 오류, 타 시스템 연동 시 오류 등 애플리케이션의 기획, 설계, 업무 시나리오 등의 단계에서 유입된 결함
GUI 결함	UI 비일관성, 데이터 타입의 표시 오류, 부정확한 커서/메시지 오류 등 사용자 화면 설계에서 발생된 결함

문서 결함

사용자의 요구사항과 기능 요구사항의 불일치로 인한 불완전한 상태의 문서, 사용자의 온라인/오프라인 매뉴얼의 불일치 등 기획자, 사용자, 개발자 간의 의사소통 및 기록이 원활하지 않아 발생한 결함

결함 심각도

결함 심각도는 애플리케이션에 발생한 결함이 전체 시스템에 미치는 치명도를 나타내는 척도이다.

- 결함 심각도를 우선순위에 따라 High, Medium, Low 또는 치명적(Critical), 주요(Major), 보통(Normal), 경미(Minor), 단순(Simple) 등으로 분류된다.

결함 우선순위

결함의 우선순위는 발견된 결함 처리에 대한 신속성을 나타내는 척도로, 결함의 중요도와 심각도에 따라 설정되고 수정 여부가 결정된다.

핵심 131 애플리케이션 성능 분석

애플리케이션 성능이란 사용자가 요구한 기능을 최소한의 자원을 사용하여 최대한 많은 기능을 신속하게 처리하는 정도를 나타낸다.

• 애플리케이션 성능 측정 지표

처리량 (Throughput)	일정 시간 내에 애플리케이션이 처리하는 일의 양
응답 시간 (Response Time)	애플리케이션에 요청을 전달한 시간부터 응답이 도착할 때까지 걸린 시간
경과 시간 (Turn Around Time)	애플리케이션에 작업을 의뢰한 시간부터 처리가 완료될 때까지 걸린 시간
자원 사용률 (Resource Usage)	애플리케이션이 의뢰한 작업을 처리하는 동안의 CPU 사용량, 메모리 사용량, 네트워크 사용량 등 자원 사용률

- 애플리케이션의 성능 분석 도구는 애플리케이션의 성능을 테스트하는 도구와 시스템을 모니터링하는 도구로 분류된다.

– 성능 테스트 도구 : 애플리케이션의 성능을 테스트하기 위해 애플리케이션에 부하나 스트레스를 가하면서 애플리케이션의 성능 측정 지표를 점검하는 도구

– 시스템 모니터링 도구 : 애플리케이션이 실행되었을 때 시스템 자원의 사용량을 확인하고 분석하는 도구



핵심 132 복잡도

복잡도(Complexity)는 시스템이나 시스템 구성 요소 또는 소프트웨어의 복잡한 정도를 나타내는 말로, 시스템 또는 소프트웨어를 어느 정도의 수준까지 테스트해야 하는지 또는 개발하는데 어느 정도의 자원이 소요되는지 예측하는데 사용된다.

시간 복잡도

시간 복잡도는 알고리즘의 실행시간, 즉 알고리즘을 수행하기 위해 프로세스가 수행하는 연산 횟수를 수치화한 것을 의미한다.

- 시간 복잡도가 낮을수록 알고리즘의 실행시간이 짧고, 높을수록 실행시간이 길어진다.
- 점근 표기법의 종류

빅오 표기법 (Big-O Notation)	<ul style="list-style-type: none"> • 알고리즘의 실행시간이 최악일 때를 표기하는 방법이다. • 입력값에 대해 알고리즘을 수행했을 때 명령어의 실행 횟수는 어떠한 경우에도 표기 수치보다 많을 수 없다.
세타 표기법 (Big-θ Notation)	<ul style="list-style-type: none"> • 알고리즘의 실행시간이 평균일 때를 표기하는 방법이다. • 입력값에 대해 알고리즘을 수행했을 때 명령어 실행 횟수의 평균적인 수치를 표기한다.
오메가 표기법 (Big-Ω Notation)	<ul style="list-style-type: none"> • 알고리즘의 실행시간이 최상일 때를 표기하는 방법이다. • 입력값에 대해 알고리즘을 수행했을 때 명령어의 실행 횟수는 어떠한 경우에도 표기 수치보다 적을 수 없다.

빅오 표기법(Big-O Notation)

빅오 표기법은 알고리즘의 실행시간이 최악일 때를 표기하는 방법으로, 신뢰성이 떨어지는 오메가 표기법이나 평가하기 까다로운 세타 표기법에 비해 성능을 예측하기 용이하여 주로 사용된다.

- 일반적인 알고리즘에 대한 최악의 시간 복잡도를 빅오 표기법으로 표현하면 다음과 같다.

$O(1)$	입력값(n)에 관계 없이 일정하게 문제 해결에 하나의 단계만을 거친다. 예) 스택의 삽입(Push), 삭제(Pop)
$O(\log_2 n)$	문제 해결에 필요한 단계가 입력값(n) 또는 조건에 의해 감소한다. 예) 이진 트리(Binary Tree), 이진 검색(Binary Search)

$O(n)$	문제 해결에 필요한 단계가 입력값(n)과 1:1의 관계를 가진다. 예) for문
$O(n \log_2 n)$	문제 해결에 필요한 단계가 $n(\log_2 n)$ 번만큼 수행된다. 예) 힙 정렬(Heap Sort), 2-Way 합병 정렬(Merge Sort)
$O(n^2)$	문제 해결에 필요한 단계가 입력값(n)의 제곱만큼 수행된다. 예) 삽입 정렬(Insertion Sort), 쉘 정렬(Shell Sort), 선택 정렬(Selection Sort), 버블 정렬(Bubble Sort), 퀵 정렬(Quick Sort)
$O(2^n)$	문제 해결에 필요한 단계가 2의 입력값(n) 제곱만큼 수행된다. 예) 피보나치 수열(Fibonacci Sequence)

순환 복잡도

순환 복잡도(Cyclomatic Complexity)는 한 프로그램의 논리적인 복잡도를 측정하기 위한 소프트웨어의 척도로, 맥케이브 순환도(McCabe's Cyclomatic) 또는 맥케이브 복잡도 메트릭(McCabe's Complexity Metrics)라고도 하며, 제어 흐름도 이론에 기초를 둔다.

- 순환 복잡도를 이용하여 계산된 값은 프로그램의 독립적인 경로의 수를 정의하고, 모든 경로가 한 번 이상 수행되었음을 보장하기 위해 행해지는 테스트 횟수의 상한선을 제공한다.
- 제어 흐름도 G에서 순환 복잡도 $V(G)$ 는 다음과 같은 방법으로 계산할 수 있다.

방법 1 순환 복잡도는 제어 흐름도의 영역 수와 일치하므로 영역 수를 계산한다.

방법 2 $V(G) = E - N + 2$: E는 화살표 수, N은 노드의 수

핵심 133 애플리케이션 성능 개선

소스 코드 최적화

소스 코드 최적화는 나쁜 코드(Bad Code)를 배제하고, 클린 코드(Clean Code)로 작성하는 것이다.

- 클린 코드(Clean Code) : 누구나 쉽게 이해하고 수정 및 추가할 수 있는 단순, 명료한 코드, 즉 잘 작성된 코드를 의미한다.
- 나쁜 코드(Bad Code)
 - 프로그램의 로직(Logic)이 복잡하고 이해하기 어려운 코드로, 스파게티 코드와 외계인 코드가 여기에 해당한다.
 - 스파게티 코드 : 코드의 로직이 서로 복잡하게 얽혀 있는 코드
 - 외계인 코드 : 아주 오래되거나 참고문서 또는 개발자가 없어 유지보수 작업이 어려운 코드
- 클린 코드 작성 원칙

가독성	<ul style="list-style-type: none"> • 누구든지 코드를 쉽게 읽을 수 있도록 작성한다. • 코드 작성 시 이해하기 쉬운 용어를 사용하거나 들여 쓰기 기능 등을 사용한다.
단순성	<ul style="list-style-type: none"> • 코드를 간단하게 작성한다. • 한 번에 한 가지를 처리하도록 코드를 작성하고 클래스/메소드/함수 등을 최소 단위로 분리한다.
의존성 배제	<ul style="list-style-type: none"> • 코드가 다른 모듈에 미치는 영향을 최소화한다. • 코드 변경 시 다른 부분에 영향이 없도록 작성한다.
중복성 최소화	<ul style="list-style-type: none"> • 코드의 중복을 최소화한다. • 중복된 코드는 삭제하고 공통된 코드를 사용한다.
추상화	상위 클래스/메소드/함수에서는 간략하게 애플리케이션의 특성을 나타내고, 상세 내용은 하위 클래스/메소드/함수에서 구현한다.

소스 코드 최적화 유형

- 클래스 분할 배치 : 하나의 클래스는 하나의 역할만 수행하도록 응집도를 높이고, 크기를 작게 작성한다.
- 느슨한 결합(Loosely Coupled) : 인터페이스 클래스를 이용하여 추상화된 자료 구조와 메소드를 구현함으로써 클래스 간의 의존성을 최소화한다.

소스 코드 품질 분석 도구

소스 코드 품질 분석 도구는 소스 코드의 코딩 스타일, 코드에 설정된 코딩 표준, 코드의 복잡도, 코드에 존재하는 메모리 누수 현상, 스레드 결함 등을 발견하기 위해

사용하는 분석 도구로, 크게 정적 분석 도구와 동적 분석 도구로 나뉜다.

- 정적 분석 도구
 - 작성한 소스 코드를 실행하지 않고 코딩 표준이나 코딩 스타일, 결함 등을 확인하는 코드 분석 도구이다.
 - 종류 : pmd, cppcheck, SonarQube, checkstyle, ccm, cobertura 등
- 동적 분석 도구
 - 작성한 소스 코드를 실행하여 코드에 존재하는 메모리 누수, 스레드 결함 등을 분석하는 도구이다.
 - 종류 : Avalanche, Valgrind 등

핵심 134 인터페이스 설계서

인터페이스 설계서는 시스템 사이의 데이터 교환 및 처리를 위해 교환 데이터 및 관련 업무, 송·수신 시스템 등에 대한 내용을 정의한 문서이다.

일반적인 인터페이스 설계서

시스템의 인터페이스 목록, 각 인터페이스의 상세 데이터 명세, 각 기능의 세부 인터페이스 정보를 정의한 문서이다.

- 시스템 인터페이스 설계서 : 시스템 인터페이스 목록을 만들고 각 인터페이스 목록에 대한 상세 데이터 명세를 정의하는 것이다.
- 상세 기능별 인터페이스 명세서
 - 각 기능의 세부 인터페이스 정보를 정의한 문서이다.
 - 인터페이스를 통한 각 세부 기능의 개요, 세부 기능이 동작하기 전에 필요한 사전/사후 조건, 인터페이스 데이터, 호출 이후 결과를 확인하기 위한 반환값 등으로 구성된다.

정적·동적 모형을 통한 인터페이스 설계서

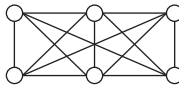
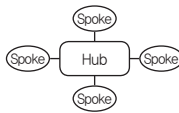
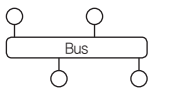
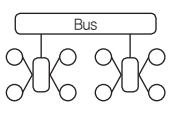
- 모형으로 각 시스템의 구성 요소를 표현한 다이어그램을 이용하여 만든 문서이다.
- 시스템을 구성하는 주요 구성 요소 간의 트랜잭션을 통해 해당 인터페이스가 시스템의 어느 부분에 속하고, 해당 인터페이스를 통해 상호 교환되는 트랜잭션의 종류를 확인할 수 있다.

핵심

135

EAI(Enterprise Application Integration)

- EAI는 기업 내 각종 애플리케이션 및 플랫폼 간의 정보 전달, 연계, 통합 등 상호 연동이 가능하게 해주는 솔루션이다.
- EAI는 비즈니스 간 통합 및 연계성을 증대시켜 효율성 및 각 시스템 간의 확정성(Determinacy)을 높여 준다.
- EAI의 구축 유형은 다음과 같다.

유형	기능
Point-to-Point	<ul style="list-style-type: none"> • 가장 기본적인 애플리케이션 통합 방식으로, 애플리케이션을 1:1로 연결한다. • 변경 및 재사용이 어렵다. 
Hub & Spoke	<ul style="list-style-type: none"> • 단일 접점인 허브 시스템을 통해 데이터를 전송하는 중앙 집중형 방식이다. • 확장 및 유지 보수가 용이하다. • 허브 장애 발생 시 시스템 전체에 영향을 미친다. 
Message Bus (ESB 방식)	<ul style="list-style-type: none"> • 애플리케이션 사이에 미들웨어를 두어 처리하는 방식이다. • 확장성이 뛰어나며 대용량 처리가 가능하다. 
Hybrid	<ul style="list-style-type: none"> • Hub & Spoke와 Message Bus의 혼합 방식이다. • 그룹 내에서는 Hub & Spoke 방식을, 그룹 간에는 Message Bus 방식을 사용한다. • 필요한 경우 한 가지 방식으로 EAI 구현이 가능하다. • 데이터 병목 현상을 최소화할 수 있다. 

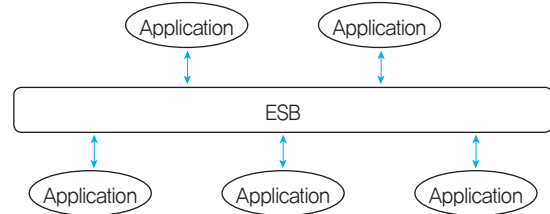
핵심

136

ESB(Enterprise Service Bus)

- ESB는 애플리케이션 간 연계, 데이터 변환, 웹 서비스 지원 등 표준 기반의 인터페이스를 제공하는 솔루션이다.
- ESB는 애플리케이션 통합 측면에서 EAI와 유사하지만 애플리케이션 보다는 서비스 중심의 통합을 지향한다.

- ESB는 특정 서비스에 국한되지 않고 범용적으로 사용하기 위하여 애플리케이션과의 결합도(Coupling)를 약하게(Loosely) 유지한다.
- 관리 및 보안 유지가 쉽고, 높은 수준의 품질 지원이 가능하다.



핵심

137

모듈 간 인터페이스 데이터 표준 확인

- 인터페이스 데이터 표준은 모듈 간 인터페이스에 사용되는 데이터의 형식을 표준화 하는 것이다.
- 인터페이스 데이터 표준은 기존의 데이터 중에서 공통 영역을 추출하거나 어느 한쪽의 데이터를 변환하여 정의한다.
- 확인된 인터페이스 데이터 표준은 인터페이스 기능 구현을 정의하는데 사용된다.

데이터 인터페이스 확인	<ul style="list-style-type: none"> • 데이터 표준을 위해 식별된 데이터 인터페이스에서 입 · 출력값의 의미와 데이터의 특성 등을 구체적으로 확인한다. • 확인된 데이터 인터페이스의 각 항목을 통해 데이터 표준을 확인한다.
인터페이스 기능 확인	<ul style="list-style-type: none"> • 데이터 표준을 위해 식별된 인터페이스 기능을 기반으로 인터페이스 기능 구현을 위해 필요한 데이터 항목을 확인한다. • 확인된 데이터 항목과 데이터 인터페이스에서 확인된 데이터 표준에서 수정 · 추가 · 삭제될 항목이 있는지 확인한다.
인터페이스 데이터 표준 확인	<ul style="list-style-type: none"> • 데이터 인터페이스에서 확인된 데이터 표준과 인터페이스 기능을 통해 확인된 데이터 항목들을 검토하여 최종적으로 데이터 표준을 확인한다. • 확인된 데이터 표준은 항목별로 데이터 인터페이스와 인터페이스 기능 중 출처를 구분하여 기록한다.

핵심 138 인터페이스 기능 구현 정의

- 인터페이스 기능 구현의 정의는 인터페이스를 실제로 구현하기 위해 인터페이스 기능에 대한 구현 방법을 기능별로 기술한 것이다.
- 인터페이스 기능 구현 정의 순서
 - ① 컴포넌트 명세서를 확인한다.
 - ② 인터페이스 명세서를 확인한다.
 - ③ 일관된 인터페이스 기능 구현을 정의한다.
 - ④ 정의된 인터페이스 기능 구현을 정형화한다.
- 모듈 세부 설계서
 - 모듈 세부 설계서는 모듈의 구성 요소와 세부적인 동작 등을 정의한 설계서이다.
 - 대표적인 모듈 세부 설계서에는 컴포넌트 명세서와 인터페이스 명세서가 있다.

컴포넌트 명세서	컴포넌트의 개요 및 내부 클래스의 동작, 인터페이스를 통해 외부와 통신하는 명세 등을 정의한 것이다.
인터페이스 명세서	컴포넌트 명세서의 항목 중 인터페이스 클래스의 세부 조건 및 기능 등을 정의한 것이다.

- 인터페이스의 기능, 인터페이스 데이터 표준, 모듈 세부 설계서를 기반으로 일관성 있고 정형화된 인터페이스 기능 구현에 대해 정의한다.
- 일관성 있는 인터페이스 기능 구현 정의
 - 인터페이스의 기능, 인터페이스 데이터 표준, 모듈 세부 설계서를 통해 인터페이스의 기능 구현을 정의한다.
 - 정의한 인터페이스 기능 구현에 대해 송·수신 측에서 진행해야 할 절차까지 다시 세부적으로 정의한다.
- 정의된 인터페이스 기능 구현 정형화
 - 정의한 인터페이스 기능 구현을 특정 하드웨어나 소프트웨어에 의존적이지 않게 사람들이 보기 쉽고 표준화되도록 정형화한다.
 - 가독성을 높이려면 프로세스 형태나 유스케이스 다이어그램 형태로 정형화한다.

핵심 139 인터페이스 구현

인터페이스 구현은 송·수신 시스템 간의 데이터 교환 및 처리를 실현해 주는 작업을 의미한다.

- 정의된 인터페이스 기능 구현을 기반으로 구현 방법 및 범위 등을 고려하여 인터페이스 구현 방법을 분석한다.

데이터 통신을 이용한 인터페이스 구현

데이터 통신을 이용한 인터페이스 구현은 애플리케이션 영역에서 인터페이스 형식에 맞춘 데이터 포맷을 인터페이스 대상으로 전송하고 이를 수신 측에서 파싱(Parsing)하여 해석하는 방식이다.

- 주로 JSON이나 XML 형식의 데이터 포맷을 사용하여 인터페이스를 구현한다.

인터페이스 엔티티를 이용한 인터페이스 구현

인터페이스 엔티티를 이용한 인터페이스 구현은 인터페이스가 필요한 시스템 사이에 별도의 인터페이스 엔티티를 두어 상호 연계하는 방식이다.

- 일반적으로 인터페이스 테이블을 엔티티로 활용한다.
- 인터페이스 테이블은 한 개 또는 송신 및 수신 인터페이스 테이블을 각각 두어 활용한다.
- 송신 및 수신 인터페이스 테이블의 구조는 대부분 같지만 상황에 따라 서로 다르게 설계할 수도 있다.

※ JSON(JavaScript Object Notation)

JSON은 속성-값 쌍(Attribute-Value Pairs)으로 이루어진 데이터 객체를 전달하기 위해 사람이 읽을 수 있는 텍스트를 사용하는 개방형 표준 포맷이다.

- 비동기 처리에 사용되는 AJAX에서 XML을 대체하여 사용되고 있다.

※ XML(eXtensible Markup Language)

XML은 특수한 목적을 갖는 마크업 언어를 만드는 데 사용되는 다목적 마크업 언어이다.

- 웹 페이지의 기본 형식인 HTML의 문법이 각 웹 브라우저에서 상호 호환적이지 못하다는 문제와 SGML의 복잡함을 해결하기 위하여 개발되었다.

※ AJAX(Asynchronous JavaScript and XML)

AJAX는 자바 스크립트(JavaScript) 등을 이용하여 클라이언트와 서버 간에 XML 데이터를 교환 및 제어함으로써 이용자가 웹 페이지와 자유롭게 상호 작용할 수 있도록 하는 비동기 통신 기술을 의미한다.



핵심 140 인터페이스 예외 처리

인터페이스 예외 처리는 구현된 인터페이스가 동작하는 과정에서 기능상 예외 상황이 발생했을 때 이를 처리하는 절차를 말한다.

- 인터페이스 예외 처리는 인터페이스를 구현하는 방법에 따라 데이터 통신을 이용한 방법과 인터페이스 엔티티를 이용한 방법이 있다.

데이터 통신을 이용한 인터페이스 예외 처리

데이터 통신을 이용한 인터페이스 예외 처리 방법은 JSON, XML 등 인터페이스 객체를 이용해 구현한 인터페이스 동작이 실패할 경우를 대비한 것이다.

- 인터페이스 객체의 송·수신 시 발생할 수 있는 예외 케이스를 정의하고 각 예외 케이스마다 예외 처리 방법을 기술한다.
- 시스템 환경, 송·수신 데이터, 프로그램 자체 원인 등 다양한 원인으로 인해 예외 상황이 발생한다.

인터페이스 엔티티를 이용한 인터페이스 예외 처리

인터페이스 엔티티를 이용한 예외 처리 방법은 인터페이스 동작이 실패할 경우를 대비하여 해당 엔티티에 인터페이스의 실패 상황과 원인 등을 기록하고, 이에 대한 조치를 취할 수 있도록 사용자 및 관리자에서 알려주는 방식으로 예외 처리 방법을 정의한다.

핵심 141 인터페이스 보안

- 인터페이스는 시스템 모듈 간 통신 및 정보 교환을 위한 통로로 사용되므로 충분한 보안 기능을 갖추지 않으면 시스템 모듈 전체에 악영향을 주는 보안 취약점이 될 수 있다.
- 인터페이스의 보안성 향상을 위해서는 인터페이스의 보안 취약점을 분석한 후 적절한 보안 기능을 적용한다.
- 인터페이스 보안 기능은 일반적으로 네트워크, 애플리케이션, 데이터베이스 영역에 적용한다.

네트워크 영역	<ul style="list-style-type: none"> • 인터페이스 송·수신 간 스니핑(Sniffing) 등을 이용한 데이터 탈취 및 변조 위협을 방지하기 위해 네트워크 트래픽에 대한 암호화를 설정한다. • 암호화는 인터페이스 아키텍처에 따라 IPSec, SSL, S-HTTP 등의 다양한 방식으로 적용한다.
애플리케이션 영역	<p>소프트웨어 개발 보안 가이드를 참조하여 애플리케이션 코드 상의 보안 취약점을 보완하는 방향으로 애플리케이션 보안 기능을 적용한다.</p>
데이터베이스 영역	<ul style="list-style-type: none"> • 데이터베이스, 스키마, 엔티티의 접근 권한과 프로시저(Procedure), 트리거(Trieger) 등 데이터베이스 동작 객체의 보안 취약점에 보안 기능을 적용한다. • 개인 정보나 업무상 민감한 데이터의 경우 암호화나 익명화 등 데이터 자체의 보안 방안도 고려한다.

핵심 142 데이터 무결성 검사 도구

- 데이터 무결성 검사 도구는 시스템 파일의 변경 유무를 확인하고, 파일이 변경되었을 경우 이를 관리자에게 알려주는 도구로, 인터페이스 보안 취약점을 분석하는데 사용된다.
- 크래커나 허가받지 않은 내부 사용자들이 시스템에 침입하면 백도어를 만들어 놓거나 시스템 파일을 변경하여 자신의 흔적을 감추는데, 무결성 검사 도구를 이용하여 이를 감지할 수 있다.
- 해시(Hash) 함수를 이용하여 현재 파일 및 디렉토리의 상태를 DB에 저장한 후 감시하다가 현재 상태와 DB의 상태가 달라지면 관리자에게 변경 사실을 알려준다.
- 대표적인 데이터 무결성 검사 도구에는 Tripwire, AIDE, Samhain, Claymore, Slipwire, Fcheck 등이 있다.

핵심 143 연계 테스트

연계 테스트는 구축된 연계 시스템과 연계 시스템의 구성 요소가 정상적으로 동작하는지 확인하는 활동이다.

연계 테스트 케이스 작성	<ul style="list-style-type: none"> 연계 시스템 간의 데이터 및 프로세스의 흐름을 분석하여 필요한 테스트 항목을 도출하는 과정이다. 송·수신용 연계 응용 프로그램의 단위 테스트 케이스와 연계 테스트 케이스를 각각 작성한다.
연계 테스트 환경 구축	테스트의 일정, 방법, 절차, 소요 시간 등을 송·수신 기관과의 협의를 통해 결정하는 것이다.
연계 테스트 수행	연계 응용 프로그램을 실행하여 연계 테스트 케이스의 시험 항목 및 처리 절차 등을 실제로 진행하는 것이다.
연계 테스트 수행 결과 검증	연계 테스트 케이스의 시험 항목 및 처리 절차를 수행한 결과가 예상 결과와 동일한지를 확인하는 것이다.

핵심 144 인터페이스 구현 검증 도구

- 인터페이스 구현을 검증하기 위해서는 인터페이스 단위 기능과 시나리오 등을 기반으로 하는 통합 테스트가 필요하다.
- 통합 테스트는 다음과 같은 테스트 자동화 도구를 이용하면 효율적으로 수행할 수 있다.

도구	기능
xUnit	Java(Junit), C++(Cppunit), .Net(Nunit) 등 다양한 언어를 지원하는 단위 테스트 프레임워크이다.
STAF	<ul style="list-style-type: none"> 서비스 호출 및 컴포넌트 재사용 등 다양한 환경을 지원하는 테스트 프레임워크이다. 크로스 플랫폼, 분산 소프트웨어 테스트 환경을 조성할 수 있도록 지원한다. 분산 소프트웨어의 경우 각 분산 환경에 설치된 데몬이 프로그램 테스트에 대한 응답을 대신하며, 테스트가 완료되면 이를 통합하고 자동화하여 프로그램을 완성한다.
FitNesse	웹 기반 테스트케이스 설계, 실행, 결과 확인 등을 지원하는 테스트 프레임워크이다.
NATF	FitNesse의 장점인 협업 기능과 STAF의 장점인 재사용 및 확장성을 통합한 NHN(Naver)의 테스트 자동화 프레임워크이다.

Selenium	다양한 브라우저 및 개발 언어를 지원하는 웹 애플리케이션 테스트 프레임워크이다.
watir	Ruby를 사용하는 애플리케이션 테스트 프레임워크이다.

핵심 145 인터페이스 구현 감시 도구

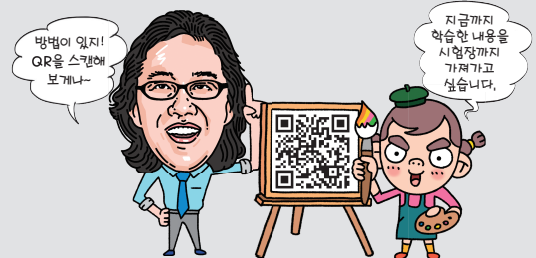
- 인터페이스 동작 상태는 APM을 사용하여 감시(Monitoring)할 수 있다.
- 애플리케이션 성능 관리 도구를 통해 데이터베이스와 웹 애플리케이션의 트랜잭션, 변수값, 호출 함수, 로그 및 시스템 부하 등 종합적인 정보를 조회하고 분석할 수 있다.
- 대표적인 애플리케이션 성능 관리 도구에는 스카우터(Scouter), 제니퍼(Jennifer) 등이 있다.

※ APM(Application Performance Management/Monitoring)

- APM은 애플리케이션의 성능 관리를 위해 접속자, 자원 현황, 트랜잭션 수행 내역, 장애 진단 등 다양한 모니터링 기능을 제공하는 도구를 의미한다.
- APM은 리소스 방식과 엔드투엔드(End-to-End)의 두 가지 유형이 있다.
 - 리소스 방식 : Nagios, Zabbix, Cacti 등
 - 엔드투엔드 방식 : VisualVM, 제니퍼, 스카우터 등

불합격 방지용 안전장치 기억상자

틀린 문제만 모아 오답 노트를 만들고 싶다고요? 까먹기 전에 다시 한 번 복습하고 싶다고요? 지금까지 공부한 내용을 안전하게 시험장까지 가져가는 완벽한 방법이 있습니다. 지금 당장 QR 코드를 스캔해 보세요.



www.membox.co.kr을 직접 입력해도 접속할 수 있습니다.